ENCICLOPEDIA PRACTICA DE LA

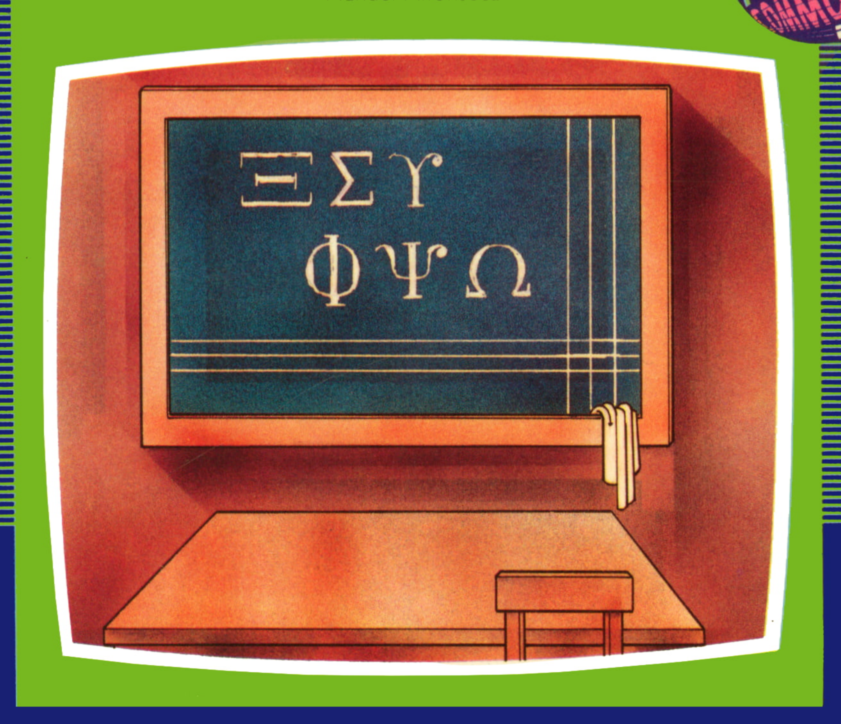
INFORMATICA APLICADA

30

Aprenda matemáticas y estadística con el lenguaje APL

SINCLAIR AMSTRAD COMMODORE IBM

Manuel Alfonseca



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

30

Aprenda matemáticas y estadística con el lenguaje APL

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo XXX. **Aprenda matemáticas y estadística con el lenguaje APL**
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta (Iber a Mart-I). Teléf.: 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-113-4
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 15.122-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta (Iberia Mart-I). Teléf.: 810 52 13. 28020 Madrid.

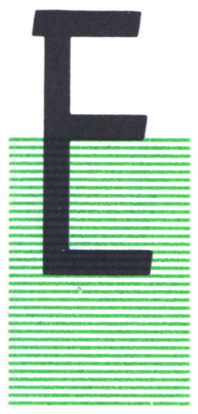
Junio, 1987
P.V.P. Canarias: 365,-

INDICE

Los programas que aparecen en este libro funcionan en los ordenadores:

IBM–PC, XT, AT y compatibles.
SINCLAIR QL.
APPLE MacINTOSH.
COMMODORE AMIGA.
ATARI 520, 1040., etc.

INTRODUCCION

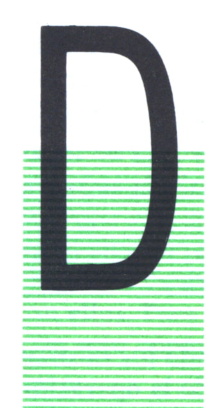


EN el número 12 de esta colección (*APL, lenguaje para programadores diferentes,* por Juan Ruiz de Torres) se ha descrito con detalle la estructura del lenguaje APL, pero sin entrar en sus posibles aplicaciones, que son infinitas, puesto que se trata de un lenguaje de uso general. Por otra parte, en el número 10 de dicha colección (*Practique matemáticas y estadística con el ordenador,* por Jesús Salcedo) se han descrito diversos programas que resuelven problemas en estas dos ciencias, utilizando para ello el lenguaje BASIC.

Ocurre que BASIC es un lenguaje poco potente, con una sintaxis muy pobre, que se mantiene en primera línea por razones principalmente históricas, pues era precisamente su pobreza lo que hacía posible construir intérpretes muy pequeños que cabían en la reducida capacidad de memoria de los microordenadores primitivos. Sin embargo, las memorias de ordenador son cada vez más baratas (ya se pueden obtener decenas de miles de posiciones por menos de diez mil pesetas) y la restricción anterior ha dejado de tener sentido. Por tanto, es de esperar que el lenguaje BASIC vaya perdiendo terreno progresivamente en favor de lenguajes más completos y estructurados (como PASCAL), o más potentes y adaptados a las técnicas de programación de la quinta generación de ordenadores, como APL o algún otro lenguaje aún más moderno que todavía no ha alcanzado la forma definitiva.

Por esta razón, y para mostrar la potencia del lenguaje APL en comparación con la de BASIC, voy a dedicar este libro a explicar cómo se programarían en aquel lenguaje diversas aplicaciones relacionadas con las matemáticas y la estadística y cómo podría utilizarse para la enseñanza de estas dos ciencias, cosa que ya se viene haciendo desde hace años en diversos países, como el autor de este libro pudo observar en una escuela de bachillerato de Sendai (Japón) en 1984.

TEORIA DE CONJUNTOS 1



ESDE hace menos de veinte años la teoría de conjuntos ha invadido la enseñanza básica española. Por alguna razón, que no puedo comprender, los responsables de la planificación educativa llegaron a la conclusión, al comienzo de la década de los setenta, de que era necesario explicar todas las matemáticas basándose únicamente en la teoría de conjuntos y abandonando por completo explicaciones más sencillas y más clásicas, basadas en los conceptos elementales de otras ramas de la misma ciencia, como la teoría de números o la combinatoria. El resultado ha sido una complicación de la enseñanza de las matemáticas que resulta, para un conocedor del tema, totalmente innecesaria y terriblemente perniciosa.

Lo más triste de todo es que la teoría de conjuntos es muy sencilla para una mentalidad ya formada. Un estudiante de los primeros cursos de la Universidad o de los últimos del bachillerato podría comprenderla por completo en un par de días de clase. Es la obsesión por enseñársela a los niños de cinco años, que aun no están preparados para ella, lo que ha convertido a las matemáticas en una de las dos bestias negras de la enseñanza elemental española (la otra es la lengua española, explicada desde la más tierna infancia con la nomenclatura de la lingüística generativa, ciencia especializada que se desarrolló durante los años cincuenta y que también debería reservarse para el nivel universitario).

La palabra conjunto es indefinible, pues cualquier definición que se intentara construir se vería obligada a recurrir a la propia palabra conjunto o a uno de sus muchos sinónimos (colección, pluralidad, reunión, etc.), por lo que no sería una definición válida. Ya se sabe que en las buenas definiciones no debe aparecer el nombre de lo definido ni tampoco ninguno de sus sinónimos. Sin embargo, no es necesario dar una definición de conjunto, pues todos tenemos un conocimiento implícito de lo que significa.

Si no nos importa el orden, una serie de números es un conjunto. En

el lenguaje APL, los conjuntos finitos de números pueden definirse por simple enumeración de sus elementos. Veamos un ejemplo:

```
N ← 1 2 3 4 5 6 7 8 9
```

Programa 1.

NOTA: Recuérdese que estamos en el teclado de un ordenador, y que debemos presionar la tecla ENTER al final de cada línea.

La expresión anterior define el conjunto de los números del uno al nueve. Como el orden no importa, este mismo conjunto podría haberse definido de !9=362880 maneras diferentes, todas ellas equivalentes entre sí. El símbolo ! representa la operación «factorial» y !9 es el número de permutaciones posibles de nueve términos. Volveremos sobre esto más adelante. Entre tanto, veamos algunas definiciones equivalentes de N, en APL:

```
N ← 9 8 7 6 5 4 3 2 1
N ← 2 4 6 8 1 3 5 7 9
N ← ⍳ 9
```

Programa 2.

En las dos primeras definiciones alternativas, hemos dado los mismos valores en un orden diferente. En la tercera hemos utilizado la operación APL representada por la letra griega «iota», que aplicada a un número entero positivo genera todos los números comprendidos entre la unidad y dicho entero. Por tanto, esta última definición es totalmente equivalente, incluso en el orden, a la que asigna a N los valores 1 2 3 4 5 6 7 8 9.

En cambio, la expresión siguiente define un conjunto totalmente diferente, al que, por tanto, asignaremos un nombre distinto. Por ejemplo, M.

```
M ← 1 3 5 7 9
```

Programa 3.

Los elementos de un conjunto pueden no ser números. En la vida real podemos encontrarnos conjuntos de manzanas o de automóviles. En un ordenador, sin embargo, el campo queda un poco restringido. Además de números, tan sólo podemos tener letras o, en general, cualquier tipo de caracteres. Como en el caso siguiente, que define el conjunto de todas las letras mayúsculas:

```
LETRAS ← 'ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Programa 4.

Obsérvense tres cosas en este conjunto: primero, que aunque hemos dado las letras en un orden determinado (el alfabético), el mismo conjunto podría haberse definido dando las mismas letras en cualquier otro or-

den. Segundo, que el nombre de un conjunto puede tener más de una letra. En este caso tiene seis. Tercero, que al revés de los conjuntos de números, cuyos elementos se enumeran separados por espacios en blanco, los caracteres se enumeran juntos, unos detrás de otros, y encerrados entre comillas.

Veamos más ejemplos de conjuntos de caracteres:

```
C ← 'aAαbBβ'
CIFRAS ← '123456789'
```

Programa 5.

El conjunto C contendrá las letras *a* y *b* en sus formas latina mayúscula y minúscula y en su forma griega (alfa y beta). En cuanto al conjunto CIFRAS, es muy peculiar. Sus elementos son los caracteres que representan las cifras del uno al nueve. Pero este conjunto no es igual, ni mucho menos, que el conjunto N, que contenía los nueve números representados por dichas cifras. En un caso tenemos números, en el otro, caracteres.

¡Bien! Ya sabemos definir conjuntos. Ahora tenemos que hacer algo con ellos. Para esto vamos a definir una serie de operaciones que afectan a un solo conjunto, a un conjunto y un elemento, o a dos conjuntos.



ELEMENTOS DE UN CONJUNTO

Supongamos que hemos definido algunos conjuntos, como los del apartado anterior. Nos puede interesar, más adelante, saber cuáles son sus elementos, tal vez porque lo hemos olvidado. Para ello, en APL bastará con nombrar el conjunto y presionar la tecla ENTER. Veamos algunos ejemplos:

```
      N
1 2 3 4 5 6 7 8 9
      M
1 3 5 7 9
      LETRAS
ABCDEFGHIJKLMNOPQRSTUVWXYZ
      C
aAαbBβ
      CIFRAS
123456789
```

Programa 6.

Obsérvese que, en el caso de los conjuntos de números, los elementos aparecen separados por espacios en blanco, cosa que no ocurre en los con-

juntos de caracteres, donde tampoco aparecen las comillas. Ahora se ve con claridad la diferencia entre los conjuntos N y CIFRAS.



CARDINAL DE UN CONJUNTO

Una de las primeras cuestiones que podemos plantearnos sobre un conjunto es la siguiente: ¿cuántos elementos tiene? Responder a esta pregunta parece muy sencillo: bastará con contarlos. Sin embargo, si el conjunto tiene quinientos elementos, resultaría un poco pesado y propenso a errores tener que contarlos uno por uno.

En APL podemos saber muy fácilmente cuántos elementos tiene un conjunto (lo que se llama el "cardinal" del conjunto). Para ello, basta con aplicarle al nombre del conjunto una operación representada por la letra griega «rho». Veamos algunos ejemplos:

```
      ρ N
9
      ρ M
5
      ρ LETRAS
26
      ρ C
6
      ρ CIFRAS
9
```

Programa 7.

Podemos observar que el cardinal de un conjunto de números es el número de números que contiene. En cambio, el cardinal de un conjunto de caracteres es el número de caracteres que aparecen entre comillas en su definición, incluido el espacio en blanco, si aparece entre las dos comillas. Veamos un ejemplo:

```
      OTRO ← 'murcielago xyz'
      OTRO
murcielago xyz
      ρ OTRO
14
```

Programa 8.

Un conjunto puede no tener ningún elemento (su cardinal es cero). Se dice entonces que está vacío. Como una cadena de caracteres (tal como 'ABCD') es un conjunto de los cuatro elementos comprendidos entre las

dos comillas, dos comillas consecutivas (") representan un conjunto de cero caracteres, es decir, un conjunto vacío.



PERTENENCIA

Una de las operaciones fundamentales que se pueden realizar con los conjuntos es averiguar si un elemento determinado (un número o un carácter) pertenece a ese conjunto o, por el contrario, si está ausente del mismo. En APL esta operación se representa con la letra griega «épsilon», a cuya izquierda se coloca el elemento del que se quiere averiguar si pertenece o no al conjunto, mientras a su derecha se coloca el nombre del conjunto. El resultado de la operación es un uno si el elemento pertenece al conjunto, y un cero en caso contrario.

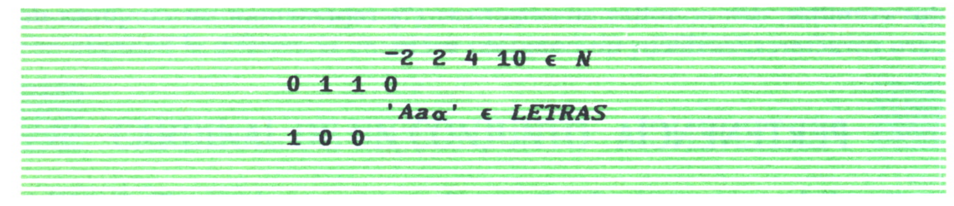
Veamos algunos ejemplos:

```
      2 ∊ N
1
      2 ∊ M
0
      2 ∊ CIFRAS
0
      '2' ∊ CIFRAS
1
      'A' ∊ N
0
      'A' ∊ LETRAS
1
      'A' ∊ OTRO
0
      'a' ∊ OTRO
1
```

Programa 9.

Obsérvese que es posible preguntar si un carácter pertenece a un conjunto de números o si un número pertenece a un conjunto de caracteres. En ambos casos, la respuesta será siempre negativa (cero). Puede verse también que las letras mayúsculas y minúsculas son realmente caracteres distintos: una de ellas (la «a») puede pertenecer a un conjunto, mientras la otra (la «A») no pertenece.

Es posible realizar la operación de pertenencia con varios elementos a la vez. En tal caso, se obtienen tantos ceros o unos como elementos hayamos colocado a la izquierda del símbolo de pertenencia. Lo único que no podemos hacer es mezclar números con caracteres. Veamos algunos ejemplos:



```
      ¯2 2 4 10 ∊ N
0 1 1 0
      'Aaα' ∊ LETRAS
1 0 0
```

Programa 10.

Por último, también podemos colocar un conjunto (en lugar de uno o varios elementos) a la izquierda del símbolo de pertenencia. En tal caso, la operación nos dice si cada uno de los elementos del conjunto de la izquierda pertenece o no al conjunto de la derecha.



```
      M ∊ N
1 1 1 1 1
      N ∊ M
1 0 1 0 1 0 1 0 1
      N ∊ LETRAS
0 0 0 0 0 0 0 0 0
      C ∊ LETRAS
0 1 0 0 1 0
      N ∊ CIFRAS
0 0 0 0 0 0 0 0 0
```

Programa 11.

Las expresiones anteriores nos dicen que todos los elementos de **M** pertenecen a **N**, pero no al revés. También vemos que ningún elemento de **N** pertenece a **LETRAS** o a **CIFRAS**, como era de esperar, puesto que un conjunto contiene números y el otro caracteres.



NO PERTENENCIA

A veces no nos interesa preguntar si un elemento pertenece a un conjunto, sino si no pertenece. Es decir, queremos hacer una pregunta que conteste afirmativamente (con un 1) si el elemento no pertenece al conjunto y negativamente (con un 0) en caso contrario. En teoría de conjuntos existe un símbolo especial para esto, que es igual que el de pertenencia, pero tachado. En APL no es necesario crear un símbolo nuevo, porque disponemos de la operación de «negación lógica», que transforma los unos en ceros y viceversa. Por tanto, la no pertenencia de un elemento a un conjunto se obtendrá fácilmente en función de la pertenencia, como en los ejemplos siguientes:

```
          ~ 2 ∊ N
0
          ~ 2 ∊ M
1
          ~ 2 ∊ CIFRAS
1
          ~ '2' ∊ CIFRAS
0
          ~ C ∊ LETRAS
1 0 1 1 0 1
```

Programa 12.

Obsérvese que la no pertenencia da siempre un resultado opuesto a la pertenencia y que también puede aplicarse entre dos conjuntos, diciéndonos en este caso si cada uno de los elementos del conjunto de la izquierda no pertenece (resultado 1) o sí pertenece (resultado 0) al conjunto de la derecha.



EXTRACCION SELECTIVA DE ELEMENTOS DE UN CONJUNTO

La operación APL representada por el símbolo «/» (una barra inclinada) actúa de la siguiente manera: a su izquierda se coloca una sucesión de ceros y unos, y a su derecha un conjunto, con la restricción de que el número de ceros y unos que aparezcan a la izquierda debe ser exactamente igual al número de elementos (al cardinal) del conjunto de la derecha. Pues bien: el resultado de la operación es otro conjunto, donde sólo aparecen los elementos del primer conjunto que corresponden a los unos de la izquierda, pero han desaparecido los que corresponden a los ceros. Veamos algunos ejemplos:

```
      1 0 1 0 1 0 1 0 1 / N
1 3 5 7 9
      0 1 0 1 0 1 0 1 0 / N
2 4 6 8
      1 1 0 1 1 0 / C
aAbB
      0 0 0 0 0 0 / C

      1 1 1 1 1 1 / C
aAαbBβ
```

Programa 13.

Obsérvese que si el operando izquierdo está formado sólo por ceros, el conjunto resultante de la operación es vacío (y aparece como una línea en blanco), mientras que si el operando izquierdo está formado sólo por unos, el resultado de la operación es el propio conjunto al que se aplica.

Hay una excepción a la regla de que el número de ceros y unos del operando izquierdo debe ser igual al cardinal del conjunto derecho: si el operando izquierdo es un solo cero, el resultado será siempre el conjunto vacío, independientemente del número de elementos del conjunto derecho. Si el operando izquierdo es un solo uno, el resultado será igual al conjunto derecho. Veámoslo:

```
      0 / C

      1 / C
aAαbBβ
```

Programa 14.



INTERSECCION DE CONJUNTOS

La intersección de dos conjuntos es un tercer conjunto que contiene únicamente los elementos que pertenecen, a la vez, a los dos conjuntos de partida. El lenguaje APL no posee un símbolo para realizar la operación intersección, pero en realidad no le hace falta, pues es trivial construir un programa que la lleve a cabo. Por ejemplo, el siguiente:

```
[0]   Z←A INTERS B
[1]   Z←(A∊B)/A
```

Programa 15.

Veamos cómo se interpreta este programa. La línea cero (señalada [0]) es la «cabecera» del programa. Indica que el nombre del programa es INTERS, que éste actúa sobre dos operandos, uno situado a su izquierda, que en la definición del programa llamamos A, y el otro situado a su derecha y que llamamos B. Finalmente, la flecha de asignación indica que la operación tiene un resultado, que dentro del programa llamaremos Z.

La línea [1], única línea ejecutable del programa, define cómo se calcula Z en función de A y de B. Supondremos que A y B son dos conjuntos, y que en Z queremos obtener su intersección. Ya hemos visto en apartados anteriores que la expresión

```
A∊B
```

Programa 16.

nos dice qué elementos del conjunto A pertenecen al conjunto B. También hemos visto que la operación representada por el símbolo «/» extrae los elementos de un conjunto que corresponden a los unos colocados a su izquierda. Por lo tanto, la expresión

```
(A∊B)/A
```

Programa 17.

nos dará la lista de los elementos de A que pertenecen a B. Es decir, la intersección de los dos conjuntos A y B.

Veamos algunos ejemplos de la aplicación de esta operación entre los conjuntos que tenemos definidos:

```
      M INTERS N
1 3 5 7 9
      N INTERS M
1 3 5 7 9
      C INTERS LETRAS
AB
      C INTERS CIFRAS

      C INTERS C
aAαbBβ
      M INTERS M
1 3 5 7 9
```

Programa 18.

Se observará que la intersección de un conjunto consigo mismo es el propio conjunto, mientras que la intersección de dos conjuntos que no tienen elementos comunes es el conjunto vacío (denotado por una línea en blanco). También puede verse que la intersección de dos conjuntos es conmutativa. Es decir, que A INTERS B es siempre igual que B INTERS A.



DIFERENCIA DE DOS CONJUNTOS

La diferencia de dos conjuntos es un tercer conjunto que contiene todos los elementos del primero que no están en el segundo. En APL construir un programa que obtenga la diferencia de dos conjuntos es una operación tan trivial como la de la intersección. Veamos cuál sería:

```
[0]   Z←A DIFER B
[1]   Z←(∼A∈B)/A
```

Programa 19.

El programa es idéntico a la intersección, salvo que aquí usamos la no pertenencia en lugar de la pertenencia, pues no nos interesan los elementos de A que pertenecen a B, sino los que no pertenecen. Veamos algunos ejemplos de su ejecución:

```
      N DIFER M
2 4 6 8
      M DIFER N

      C DIFER LETRAS
aαbβ
      LETRAS DIFER C
CDEFGHIJKLMNOPQRSTUVWXYZ
      C DIFER N
aAαbBβ
      N DIFER C
1 2 3 4 5 6 7 8 9
```

Programa 20.

Obsérvese que cuando un conjunto está comprendido totalmente en otro (como M respecto a N) su diferencia es vacía. Por otra parte, cuando dos conjuntos no tienen elementos comunes, su diferencia es igual al conjunto de la izquierda. Puede verse también que la operación diferencia de conjuntos no es conmutativa, pues A DIFER B es distinto, en general, de B DIFER A.



UNION DE CONJUNTOS

La unión de dos conjuntos es un tercer conjunto que contiene todos los elementos que se encuentran al menos en uno de los dos conjuntos. Si un elemento se encuentra en los dos conjuntos, aparecerá una sola vez en la unión de ambos. Es decir, la unión junta los elementos, pero sin repetirlos.

Existe una operación APL que junta los elementos de dos conjuntos, pero sin eliminar los repetidos. Esta operación se representa mediante una coma. Veamos algún ejemplo de su funcionamiento:

```
      N,M
1 2 3 4 5 6 7 8 9 1 3 5 7 9
      M,N
1 3 5 7 9 1 2 3 4 5 6 7 8 9
      C,CIFRAS
aAαbBβ123456789
```

Programa 21.

No pueden unirse, en APL, conjuntos de números con conjuntos de caracteres. Si se intenta realizar esta operación, se producirá un mensaje de error. La única excepción a esta regla es que el conjunto vacío puede unirse tanto con conjuntos de números como con conjuntos de caracteres.

Es fácil comprobar que la verdadera unión de dos conjuntos (la unión sin repetición) es siempre igual a la unión del primero de los dos conjuntos con la diferencia del segundo respecto al primero. En efecto, dicha diferencia contiene precisamente los elementos que faltaban para completar la unión: los del segundo conjunto que no estaban en el primero. Veamos, por tanto, una función APL que calcula la unión de dos conjuntos:

```
[0]   Z←A UNION B
[1]   Z←A,(B DIFER A)
```

Programa 22.

Veamos también algunos ejemplos de su utilización:

```
        N UNION M
1 2 3 4 5 6 7 8 9
        M UNION N
1 3 5 7 9 2 4 6 8
        C UNION LETRAS
aAαbBβCDEFGHIJKLMNOPQRSTUVWXYZ
        LETRAS UNION C
ABCDEFGHIJKLMNOPQRSTUVWXYZaαbβ
        M UNION ''
1 3 5 7 9
```

Programa 23.

En teoría de conjuntos, el orden de los elementos no importa, por lo que los resultados de N UNION M y de M UNION N son equivalentes. Por tanto, la unión de conjuntos tiene la propiedad conmutativa. Véase además que la unión de dos conjuntos puede ser igual a uno de ellos, si el otro es vacío o está totalmente incluido en él, como en el caso de M y N.



SUBCONJUNTOS

Se dice que un conjunto A es subconjunto de otro conjunto B, cuando todos los elementos del primero están también en el segundo. En teoría de conjuntos esto se expresa de la siguiente forma:

```
A ⊂ B
```

Programa 24.

Sin embargo, podemos darnos cuenta de que siempre que un conjunto A es subconjunto de otro conjunto B, la diferencia entre A y B es el conjunto vacío. Por tanto, en APL podemos construir un programa que nos diga si un conjunto es subconjunto de otro (devolverá el resultado 1) o si no lo es (devolverá un 0) haciendo uso de esta propiedad.

```
[0]    Z←A SUBC B
[1]    Z←0=ρA DIFER B
```

Programa 25.

Este programa produce un resultado igual a 1 cuando el número de elementos (el cardinal) de la diferencia de los dos conjuntos en el orden ade-

cuado es igual a cero, es decir, cuando la diferencia es el conjunto vacío. Veamos algunos ejemplos:

```
          N SUBC M
0
          M SUBC N
1
          C SUBC LETRAS
0
          'ABCD' SUBC LETRAS
1
          M SUBC M
1
          C SUBC C
1
          '' SUBC N
1
          '' SUBC LETRAS
1
```

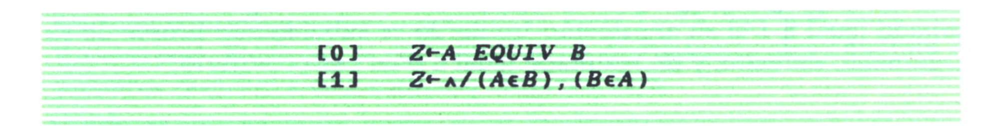
Programa 26.

Podemos ver en los ejemplos que la operación subconjunto no es conmutativa, pues M es subconjunto de N, pero no al revés. Además, es fácil comprobar que todo conjunto es subconjunto de sí mismo y que el conjunto vacío (denotado por dos comillas consecutivas) es subconjunto de cualquier otro.



EQUIVALENCIA DE CONJUNTOS

Dos conjuntos son equivalentes cuando tienen los mismos elementos, aunque pueden estar en distinto orden. La siguiente función APL comprueba si dos conjuntos son equivalentes y obtiene un resultado igual a 1 si lo son, y a 0 si no lo son:



```
[0]   Z←A EQUIV B
[1]   Z←∧/(A∊B),(B∊A)
```

Programa 27.

Esta función obtiene primero la lista de ceros y unos de los elementos de B que pertenecen a A; después le concatena por la izquierda la lista de

ceros y unos de los elementos de A que pertenecen a B. Los dos conjuntos serán equivalentes si, y sólo si la lista resultante sólo tiene unos. La reducción (∧/) da un resultado igual a 1 si todo lo que tiene a su derecha son unos, y un resultado igual a 0 en cuanto exista un cero en la lista, es decir, en cuanto un elemento de A no pertenezca a B o viceversa. Luego esta función nos dará un 1 si los dos conjuntos son equivalentes y un 0 en caso contrario. Veamos algunos ejemplos:

```
      N EQUIV M
0
      N EQUIV 1 8 7 6 4 5 3 2 9
1
      N EQUIV (N UNION M)
1
      M EQUIV (N UNION M)
0
      (C UNION LETRAS) EQUIV (LETRAS UNION C)
1
      N EQUIV CIFRAS
0
```

Programa 28.

Puede verse que la unión tiene la propiedad conmutativa (pues C UNION LETRAS es equivalente a LETRAS UNION C) y que el orden de los elementos no importa para que los dos conjuntos sean equivalentes. Es posible comparar con esta operación conjuntos de letras con conjuntos de caracteres (como N y CIFRAS), pero, como es lógico, el resultado será siempre cero.



SUBCONJUNTOS PROPIOS

Se dice que un conjunto A es subconjunto propio de otro conjunto B cuando es subconjunto de él, pero los dos conjuntos no son equivalentes. Veamos el programa APL que lo comprueba, es decir, que nos da un resultado igual a 1 si su argumento izquierdo es subconjunto propio de su argumento derecho, y un resultado igual a 0 en caso contrario:

```
[0]   Z←A SUBCP B
[1]   Z←(A SUBC B)∧~(A EQUIV B)
```

Programa 29.

donde la tilde (~) representa la operación «no», que cambia los ceros por unos y los unos por ceros, y el símbolo ∧ representa la operación «y», que da un resultado de 1 si sus dos argumentos (izquierdo y derecho) son ambos iguales a 1 y un 0 en caso contrario. Por tanto, la línea [1] de la función SUBCP puede leerse: «A es subconjunto propio de B si A es subconjunto de B y no es A equivalente a B». Veamos algunos ejemplos:

```
            M SUBCP N
1
            N SUBCP N
0
            'ABCD' SUBCP LETRAS
1
            M SUBCP M
0
            C SUBCP C
0
            '' SUBCP N
1
            '' SUBCP LETRAS
1
```

Programa 30.

Se observará que ningún conjunto es subconjunto propio de sí mismo, y que el conjunto vacío lo es de cualquier otro conjunto, sea de números o de caracteres, excepto de sí mismo.



CONJUNTOS COMPLEMENTARIOS

Si tenemos un conjunto universal determinado, llamaremos conjunto complementario de un conjunto dado A, respecto de dicho conjunto universal, al conjunto de todos los elementos del conjunto universal que no pertenecen a A. Después de todo lo que llevamos visto, el programa APL que obtiene el complementario de un conjunto dado respecto a cierto conjunto universal es trivial. Por ejemplo, supongamos que vamos a trabajar sólo con números enteros comprendidos entre 1 y 25. Este será, pues, nuestro universo. Entonces, el complementario de cualquier otro conjunto de números respecto a este conjunto universal será:

```
[0]   Z←COMPL25 A
[1]   Z←(⍳25) DIFER A
```

Programa 31.

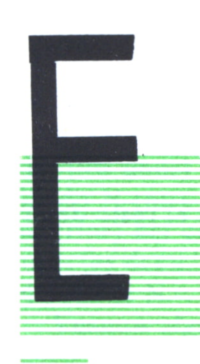
pues, efectivamente, el conjunto complementario de uno dado no es otra cosa que la diferencia entre el conjunto universal y dicho conjunto. Veamos algunos ejemplos:

```
      COMPL25 N
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
      COMPL25 M
2 4 6 8 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
      COMPL25 1 3 5 7 9 11 13 15 17 19 21 23 25
2 4 6 8 10 12 14 16 18 20 22 24
      N1←1 3 5 7 9 11 13 15 17
      N2←2 4 6 8 10 12 14 15 16 17
      (COMPL25 (N1 UNION N2)) EQUIV ((COMPL25 N1) INTERS (COMPL25 N2))
1
      (COMPL25 (N1 INTERS N2)) EQUIV ((COMPL25 N1) UNION (COMPL25 N2))
1
```

Programa 32.

Los dos últimos ejemplos son dos casos particulares de las «leyes de De Morgan», que afirman que «el complementario de la unión de dos conjuntos es igual a la intersección de sus complementarios», y «el complementario de la intersección de dos conjuntos es igual a la unión de sus complementarios».

TEORIA DE NUMEROS 2



S ésta una materia amplísima, cuyas aplicaciones alcanzan los campos más insospechados (como la criptografía) y que se encuentra en primera línea de la investigación matemática contemporánea. Dentro de este capítulo veremos tan sólo algunas cuestiones relacionadas con la divisibilidad de números enteros.

RESTO DE DIVIDIR DOS NUMEROS ENTRE SI

La operación APL (A | B) da como resultado el resto de dividir B entre A. Se trata de una operación aritmética, como la suma, resta, multiplicación y división, que puede extenderse como éstas a series y tablas de datos (vectores y matrices). Téngase cuidado al utilizarla, pues el dividendo se escribe a su derecha y el divisor a su izquierda, al contrario que en la división.

Veamos algunos ejemplos de su uso:

```
      5|12
2
      12|5
5
      5|⍳10
1 2 3 4 0 1 2 3 4 0
      3|2 4 6
2 1 0
```

Programa 1.

Veamos ahora cómo puede obtenerse en APL la tabla de restos, es decir, una tabla equivalente a la de sumar, pero que, en lugar de la suma,

nos da el resto de dividir la cabecera de cada columna entre la cabecera de cada fila.

```
      (⍳9) ∘.| (⍳9)
0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 1
1 2 0 1 2 0 1 2 0
1 2 3 0 1 2 3 0 1
1 2 3 4 0 1 2 3 4
1 2 3 4 5 0 1 2 3
1 2 3 4 5 6 0 1 2
1 2 3 4 5 6 7 0 1
1 2 3 4 5 6 7 8 0
```

Programa 2.

donde el valor situado en la fila «i» y en la columna «j» es el resto de dividir «j» entre «i».



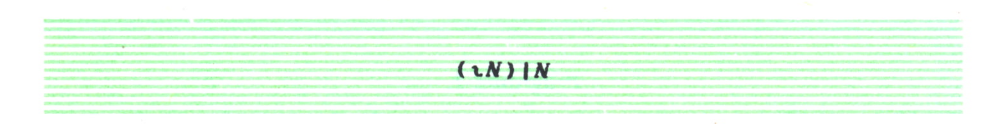
DIVISORES DE UN NUMERO

A veces puede ser interesante obtener el conjunto de los divisores de un número dado, es decir, el conjunto de todos los números enteros por los que el número dado es divisible sin dar resto alguno. En APL esto puede conseguirse mediante el siguiente programa:

```
[0]    Z←DIV N
[1]    Z←(0=(⍳N)|N)/⍳N
```

Programa 3.

Veamos cómo lo consigue. Supongamos que N fuera igual a 60. Entonces, el resultado de aplicar la letra griega «iota» al número 60 es el conjunto de todos los números consecutivos, desde 1 hasta 60. Obsérvese la siguiente expresión:



Programa 4.

que aparece en el programa. Consiste en hallar el resto de la división del número N entre todos los números enteros comprendidos entre 1 y N. Por tanto, el resultado de esta operación es el conjunto de todos los restos obtenidos. Seleccionamos ahora los que son iguales a cero (es decir, los que corresponden a los divisores de N) mediante la expresión:

```
(0=(⍳N)|N)
```

Programa 5.

El resultado de esta expresión es una serie de unos y ceros que nos dicen (los unos) cuáles son los números entre 1 y N que dan resto cero al dividir N entre ellos. Si ahora extraemos selectivamente (véase el capítulo anterior) del conjunto de números de 1 a N los que corresponden a los unos (es decir, los que dan resto cero), obtendremos directamente los divisores de N.

Veamos algunos ejemplos:

```
          DIV 60
1 2 3 4 5 6 10 12 15 20 30 60
          DIV 100
1 2 4 5 10 20 25 50 100
          DIV 37
1 37
```

Programa 6.

Obsérvese que 37 sólo puede dividirse por sí mismo y por la unidad.

Todavía podemos simplificar más el programa anterior. Obsérvese que, tal como está, este programa se ve obligado a calcular dos veces el conjunto de los números de 1 a N. Pero si la primera vez que lo calcule guardamos este conjunto en una variable, la segunda vez podremos utilizar esta variable y abstenernos de volverla a calcular. El programa modificado quedará así:

```
[0]    Z←DIV N
[1]    Z←(0=Z|N)/Z←⍳N
```

Programa 7.



MAXIMO COMUN DIVISOR DE DOS NUMEROS

Para calcular el máximo común divisor de dos números (A y B) suele emplearse el algoritmo de Euclides, que puede describirse así:

1. Obtener el resto de dividir A entre B.
2. Si dicho resto es cero, el máximo común divisor es el divisor.
3. Si no lo es, tomamos como dividendo el antiguo divisor, y como nuevo divisor el resto de la división anterior y volvemos al paso 1.

El siguiente programa APL calcula el máximo común divisor de sus dos argumentos utilizando este algoritmo:

```
[0]     Z←A MCD B
[1]   L:Z←A
[2]     A←A|B
[3]     B←Z
[4]     →(A≠0)/L
```

Programa 8.

La línea [4] de este programa es una transferencia condicional, que puede leerse así: «si el valor de A es distinto de cero, saltar a la instrucción que tiene la etiqueta L; en caso contrario, seguir». Veamos algunos ejemplos de su utilización:

```
      3 MCD 4
1
      12 MCD 16
4
      12 MCD 16 MCD 22
2
      3 MCD 9
3
```

Programa 9.

Como se ve, el mismo programa puede utilizarse para calcular el máximo común divisor de más de dos números, pues la operación «máximo común divisor» es asociativa, es decir, podemos agrupar de cualquier manera los números de los que queremos calcular el máximo común divisor y obtenerlo progresivamente de dos en dos.

El programa anterior tiene un bucle, pues utiliza el algoritmo de Euclides. También es posible obtener directamente el máximo común divi-

sor con un programa APL que no tiene ningún bucle, lo que probablemente lo hará más rápido. Hemos visto en el apartado anterior cómo se obtienen los divisores de un número. Sabemos que el máximo común divisor de dos números es el mayor de los divisores comunes a los dos. Luego podremos obtenerlo calculando todos los divisores de ambos, quedándonos sólo con los comunes y eligiendo entre éstos el mayor. Veamos el programa que lo hace así:

```
[0]   Z←A MCD B
[1]   Z←⌈/(DIV A) INTERS (DIV B)
```

Programa 10.

Este programa llama a otros dos, definidos previamente: DIV, que calcula todos los divisores de un número, e INTERS, que obtiene la intersección de dos conjuntos, y que vimos en el capítulo anterior. Puede comprobarse que esta versión de MCD produce los mismos resultados que la versión anterior, que utiliza el algoritmo de Euclides.



MINIMO COMUN MULTIPLO DE DOS NUMEROS

Como se sabe, el mínimo común múltiplo de dos números es el más pequeño de sus múltiplos comunes, es decir, el número más pequeño que puede obtenerse a partir de los dos números dados, multiplicando cada uno de ellos por algún número entero.

El mínimo común múltiplo de dos números puede calcularse dividiendo su producto por su máximo común divisor. Por tanto, es muy fácil construir un programa APL que lo obtenga:

```
[0]   Z←A MCM B
[1]   Z←(A×B)÷(A MCD B)
```

Programa 11.

Veamos algunos ejemplos:

```
          3 MCM 4
12
          12 MCM 16
48
          12 MCM 16 MCM 22
528
          3 MCM 9
9
```

Programa 12.

Obsérvese que, como la operación de calcular el mínimo común múltiplo es también asociativa, para calcular el de tres o más números basta con aplicar el programa MCM dos a dos, en cualquier orden.



NUMEROS PRIMOS

Se llama número primo al que tiene como divisores únicamente la unidad y el propio número. Se exceptúa el 1, que por convenio no se considera primo. Por tanto, un número primo puede definirse como aquél que tiene exactamente dos divisores. Con esta definición, el 1 queda automáticamente excluido, pues tiene un solo divisor (él mismo).

Como ya disponemos de un programa que nos calcula el conjunto de los divisores de un número, la comprobación de si un número es primo o no es trivial:

```
[0]   Z←PRIMO N
[1]   Z←2=⍴DIV N
```

Programa 13.

Para comprender cómo funciona este programa, hay que recordar que la operación representada por la letra griega rho devuelve el cardinal (el número de elementos) del conjunto al que se aplica (véase el capítulo primero).

Cuando aplicamos la función PRIMO a un número determinado, nos devuelve un uno si dicho número es primo y un cero en caso contrario.

Veamos algunos ejemplos de su uso:

```
          PRIMO 1
0
          PRIMO 2
1
          PRIMO 21
0
          PRIMO 37
1
```

Programa 14.

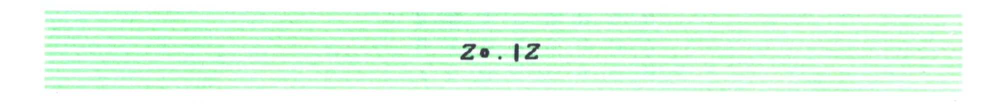
Ahora vamos a construir una función que nos calcula todos los números primos menores que uno dado:

```
[0]   Z←PRIMOS N
[1]   Z←(2=+/[1]0=Z∘.|Z)/Z←⍳N
```

Programa 15.

Veamos cómo funciona. Puesto que en APL, en ausencia de paréntesis, las operaciones se ejecutan siempre de derecha a izquierda, lo primero que se efectuará en el programa anterior es la asignación a Z del conjunto de todos los números de 1 a N, que constituye el conjunto de partida, del cual vamos a eliminar todos aquéllos que no sean primos, para quedarnos únicamente con éstos, que son los que nos interesan.

La operación



```
Z∘.|Z
```

Programa 16.

nos calcula la tabla de restos cuyas filas y columnas vienen encabezadas por los números de 1 a N, tal como hemos explicado anteriormente. Por ejemplo, supongamos que N es igual a 9. En tal caso, la tabla obtenida en este punto será la misma que vimos al principio de este capítulo, es decir:

```
0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 1
1 2 0 1 2 0 1 2 0
1 2 3 0 1 2 3 0 1
1 2 3 4 0 1 2 3 4
1 2 3 4 5 0 1 2 3
1 2 3 4 5 6 0 1 2
1 2 3 4 5 6 7 0 1
1 2 3 4 5 6 7 8 0
```

Programa 17.

La operación

```
0=Z∘.|Z
```

Programa 18.

compara todos los elementos de la tabla de restos con cero y forma una nueva tabla, donde los ceros han sido sustituidos por unos y los números distintos de cero por ceros. Los unos señalan las posiciones de la tabla que corresponden a restos iguales a cero, es decir, a divisiones exactas. En el caso de N=9, la tabla resultante quedará así:

```
1 1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1 0
0 0 1 0 0 1 0 0 1
0 0 0 1 0 0 0 1 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1
```

Programa 19.

Cada una de las columnas de esta nueva tabla nos indica, por ello, cuáles son los divisores de cada uno de los números de 1 a N. Por tanto, si sumamos por columnas los valores de esta tabla, obtendremos el número

de los divisores de cada uno de los números de 1 a N, de un solo plumazo. Esto es lo que realiza la operación siguiente, que indicamos junto con su resultado:

```
      +/[1]0=Z∘.|Z
1 2 2 3 2 4 2 4 3
```

Programa 20.

El resultado anterior nos dice que el 1 tiene un solo divisor, el 2 y el 3 tienen dos, el 4 tiene tres divisores, etc. Por último, basta con seleccionar del conjunto Z (los números de 1 a N) aquéllos que tienen exactamente dos divisores, que serán primos. Veamos algún ejemplo del uso de este programa:

```
      PRIMOS 20
2 3 5 7 11 13 17 19
```

Programa 21.



DIVISORES PRIMOS DE UN NUMERO

El programa siguiente calcula los divisores primos de un número, utilizando una combinación de los programas anteriores:

```
[0]   Z←FACT_PRIMOS N;A
[1]    A←DIV N
[2]    Z←''
[3]   L:→(~PRIMO 1↑A)/L1
[4]    Z←Z,1↑A
[5]   L1:A←1↓A
[6]    →(0≠⍴A)/L
```

Programa 22.

Esta función asigna primero a la variable local A los divisores de N. Después aplica a cada uno de ellos la función PRIMO, que le dice cuáles son primos y cuáles no lo son. Sólo los primeros pasan a formar parte del resultado Z. Veamos algunos ejemplos:

```
          FACT_PRIMOS 60
2 3 5
          FACT_PRIMOS 100
2 5
          FACT_PRIMOS 37
37
```

Programa 23.

Como es natural, el único factor primo de un número primo (como 37) es él mismo.



PROGRESIONES ARITMETICAS

Una progresión aritmética es una sucesión de números, caracterizados porque la diferencia entre dos consecutivos es siempre la misma. Por ejemplo: 1 3 5 7 9 es una progresión aritmética de diferencia 2. En cambio, 2 6 10 14 es una progresión aritmética de diferencia 4.

Una progresión aritmética queda siempre totalmente definida mediante tres números enteros: el primer término de la progresión, la diferencia entre dos términos consecutivos y el número de términos. Vamos ahora a construir un programa APL que calcula todos los términos de una progresión aritmética en función de esos tres datos:

```
[0]   Z←PROGRESION N
[1]   Z←N[1]+(N[2]×0,⍳(N[3]-1))
```

Programa 24.

Se supone que N es una serie o vector de tres elementos, que son los tres datos: primer término de la progresión, diferencia y número de términos, en ese orden. La expresión

```
⍳(N[3]-1)
```

Programa 25.

calcula todos los números entre 1 y N[3]-1. A esa serie se le añade ahora un cero por la izquierda, lo que da lugar a una serie de N[3] términos, con valores comprendidos entre 0 y N[3]-1. Ésta es ya una progresión aritmética cuyo primer término es 0, su diferencia es 1 y que tiene el número de términos deseado (N[3]). A continuación, la serie entera se multiplica por N[2]. Obtenemos así una nueva progresión artimética cuyo primer término es 0, la diferencia es N[2], y el número de términos es N[3]. Finalmente, si sumamos N[1] a todos los términos de esta progresión, obtendremos la progresión deseada.

Veamos algunos ejemplos:

```
      PROGRESION 5 3 6
5 8 11 14 17 20
      PROGRESION 1 5 5
1 6 11 16 21
      PROGRESION 20 ¯4 7
20 16 12 8 4 0 ¯4
```

Programa 26.

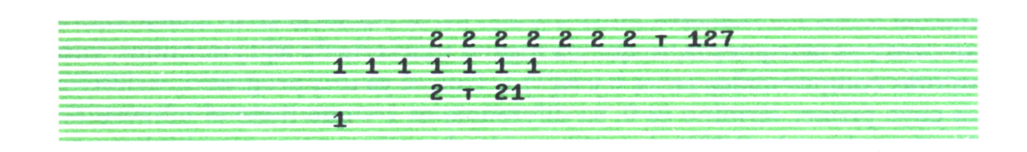


CAMBIO DE BASE DE NUMERACION

En APL es muy fácil cambiar de sistema de base de numeración, pues existen dos operaciones elementales del lenguaje que lo realizan directamente. Una de estas dos operaciones tiene la forma de una pequeña T mayúscula y sirve para convertir números de la base 10 (la que utilizamos en la vida ordinaria) a otra base cualquiera. El número a convertir se da a la derecha del símbolo que representa la operación. A la izquierda se coloca la base a la que se desea convertir, repetida tantas veces como cifras deseemos obtener del número convertido. ¡Atención! Si exigimos menos cifras de las necesarias, la operación se realizará de todos modos, pero obtendremos sólo las cifras menos significativas. En cambio, si pedimos más cifras de las necesarias, obtendremos uno o varios ceros a la izquierda.

Veamos algunos ejemplos:

```
      8 8 8 8 8 ⊤ 1000
0 1 7 5 0
      16 16 16 16 ⊤ 1000
0 3 14 8
```



```
                2 2 2 2 2 2 2 ⊤ 127
          1 1 1 1 1 1 1
                2 ⊤ 21
          1
```

Programa 27.

Obsérvese que es posible saber si un número es par o impar pasándolo a base 2, pero quedándonos sólo con la última cifra. Si ésta es 0, el número será par; si es 1, será impar. También puede verse que, si la base a la que deseamos convertir es mayor que 10, algunos de los términos obtenidos podrán tener más de una cifra. Ahora bien: en base 16, estamos acostumbrados a trabajar con letras, en lugar de números. Para obtenerlas, bastará con formar la serie de las dieciséis cifras posibles en esta base, como cadena de caracteres, que indexaremos con el resultado de la operación de cambio de base:

```
      '0123456789ABCDEF' [1 + 16 16 16 16 ⊤ 1000]
03E8
```

Programa 28.

La otra operación de cambio de base en APL pasa de una base cualquiera a la base 10 y se representa con el símbolo inverso del anterior (una pequeña T mayúscula invertida). En este caso, el argumento derecho debe ser la serie de las cifras del número en la base de partida, mientras que el argumento izquierdo es la base de partida, que en este caso puede escribirse una sola vez. El resultado es el número en base 10.

```
      8 ⊥ 1 7 5 0
1000
      16 ⊥ 3 14 8
1000
      2 ⊥ 1 1 1 1 1 1 1
127
```

Programa 29.

En las operaciones de cambio de base es posible convertir de la base 10 a más de una base de numeración al mismo tiempo y viceversa. Esto es útil para convertir cierto número de segundos de arco al sistema sexa-

gesimal de medida de ángulos (grados, minutos y segundos) y también para calcular cuántos segundos corresponden a cierto número de grados, minutos y segundos:

```
         360 60 60 ⊤ 5000
   1 23 20
         360 60 60 ⊥ 1 23 20
   5000
```

Programa 30.

Vemos que 5000 segundos de arco (5000») equivalen a 1° 23' 20''. Vemos también cómo se realiza la conversión contraria. Un procedimiento muy parecido puede emplearse para convertir segundos de tiempo a horas, minutos y segundos, y viceversa:

```
         24 60 60 ⊤ 3700
   1 1 40
         24 60 60 ⊥ 1 1 40
   3700
```

Programa 31.

Vemos así que 3700 segundos de tiempo equivalen a 1 hora, 1 minuto y 40 segundos. También vemos cómo se ha realizado la conversión contraria.

¿Cómo pasaríamos un número de la base 8 a la base 5? Muy fácil: lo pasamos primero a la base 10 desde la base de partida, y desde ésta a la base final en una segunda operación. Como ejemplo, vamos a pasar el número 3 3 2 (expresado en base 8) a la base 5:

```
         5 5 5 5 ⊤ 8 ⊥ 3 3 2
   1 3 3 3
```

Programa 32.

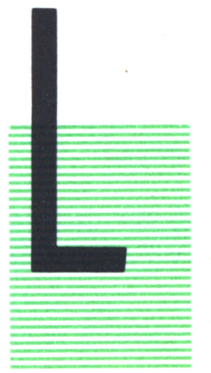
Para operar con dos números en una base distinta de 10, podemos hacer algo parecido. Supongamos que queremos sumar dos números escritos en base 3 (como 1 2 1 y 2 0). Pues bien: bastará con pasarlos a la base 10, sumarlos en ella y volver a pasarlos a la base 3 para obtener el resultado deseado:

```
            3 3 3 ⊤ (3 ⊥ 1 2 1) + (3 ⊥ 2 0)
     2 1 1
```

Programa 33.

En lugar de la suma podríamos haber realizado cualquier otra operación.

COMBINATORIA 3



LA combinatoria es la rama de las matemáticas que estudia las diversas formas en que se pueden ordenar o agrupar varios objetos. Si lo que nos interesa ante todo es el orden, obtendremos distintas «variaciones». Si no nos importa el orden, sino sólo la manera de agruparse, tenemos el problema de las «combinaciones». Ambos son muy importantes y tienen numerosas aplicaciones en la vida real y en el estudio del mundo físico.



VARIACIONES

Supongamos que tenemos un conjunto de M objetos diferentes y que queremos saber de cuántas maneras distintas se pueden disponer en grupos o subconjuntos de N elementos, donde el orden importa, es decir, consideramos que dos grupos son distintos si el orden de los elementos difiere.

Por ejemplo: sea el conjunto de las cifras del cero al nueve:

0 1 2 3 4 5 6 7 8 9

Programa 1.

¿Cuántos números de dos cifras distintas podremos construir con ellos? Es evidente que los números de dos cifras pueden considerarse como subconjuntos de dos elementos del conjunto anterior y que, en este caso, el orden de las cifras importa, pues 23 es un número diferente de 32.

Pues bien: el cálculo de variaciones nos demuestra que el número de grupos distintos de N elementos que podemos formar con un conjunto de M elementos es igual a:

```
Mx(M-1)x(M-2)x. . . x(1+M-N)
```

Programa 2.

En nuestro ejemplo, M=10, N=2. Por tanto, 1+M-N=1+10-2=9. Luego el número buscado es 10x9=90. Y los 90 números que se pueden formar con dos cifras distintas son:

```
01 02 03 04 05 06 07 08 09
10 12 13 14 15 16 17 18 19
20 21 23 24 25 26 27 28 29
30 31 32 34 35 36 37 38 39
40 41 42 43 45 46 47 48 49
50 51 52 53 54 56 57 58 59
60 61 62 63 64 65 67 68 69
70 71 72 73 74 75 76 78 79
80 81 82 83 84 85 86 87 89
90 91 92 93 94 95 96 97 98
```

Programa 3.

El programa siguiente, escrito en APL, calcula el número de variaciones que pueden construirse con un conjunto de M elementos, tomándolos en subconjuntos de N donde el orden es importante:

```
[0]    Z←M VAR N
[1]    Z←x/M-0,ιN-1
```

Programa 4.

Comprobemos que el programa anterior calcula el número de variaciones deseado. Para ello, vamos a ver lo que hace, procediendo, como es de rigor en APL, de derecha a izquierda:

1. Lo primero que encontramos en la única instrucción de que consta este programa es la expresión

```
ιN-1
```

Programa 5.

que, como ya sabemos, nos calcula los números entre 1 y N-1. Así, en el caso de nuestro ejemplo, donde N=2, N-1 es igual a 1 y, por tanto, obtendremos todos los números entre 1 y 1, es decir: sólo el 1.

2. En segundo lugar, la expresión

```
0,ιN-1
```

Programa 6.

concatena por la izquierda un cero al resultado anterior. Como éste era el conjunto de los números desde 1 hasta N-1, al añadir el cero obtendremos todos los números entre cero y N-1. En nuestro ejemplo, el par de números 0 y 1.

3. La expresión siguiente que encontramos es:

```
M-0,ιN-1
```

Programa 7.

que no hace otra cosa que restar de M cada uno de los números obtenidos anteriormente. Como éstos estaban comprendidos entre 0 y N-1, ahora obtendremos la serie M-0, M-1, ..., M-(N-1), es decir, la serie M, M-1, ..., 1+M-N, cuyo producto es igual al número de variaciones buscado.

4. Dicho producto es calculado por la expresión

```
×/M-0,ιN-1
```

Programa 8.

que nos da el resultado deseado.

Veamos algunos ejemplos:

```
      10 VAR 2
90
      10 VAR 3
720
      26 VAR 3
15600
```

Programa 9.

donde el último ejemplo nos calcula el número de palabras distintas de tres letras que podrían formarse con las 26 letras del alfabeto inglés, sin repetir ninguna letra en cada palabra.

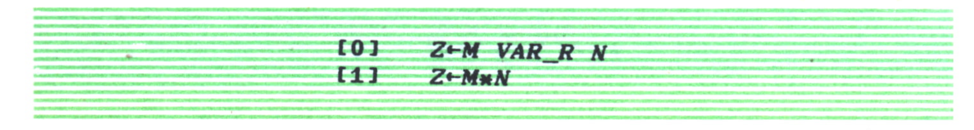


VARIACIONES CON REPETICION

El problema del cálculo de variaciones difierè ligeramente en el caso de que sea posible repetir los elementos del conjunto de partida en los grupos o variaciones que vamos formando. Se observará que los números de dos cifras obtenidos anteriormente tienen siempre las dos cifras distintas entre sí, lo que se debe a que habíamos impuesto la restricción de que los elementos no podían repetirse. Pero esto significa que los números que tienen las dos cifras iguales (00, 11, 22, 33, 44, 55, 66, 77, 88 y 99) no han aparecido en nuestra lista. Si queremos contarlos también, hemos de trabajar con las «variaciones con repetición».

Vamos a calcular el número de variaciones con repetición que se pueden formar con los M elementos de un conjunto, constando cada variación de N elementos, donde el orden importa y los elementos de M pueden repetirse. De acuerdo con el cálculo de variaciones, este número es igual a M elevado a N. En nuestro ejemplo, donde M=10 y N=2, el número total de variaciones con repetición será igual a 10 al cuadrado, es decir, 100. Y en efecto, añadiendo los 10 números de cifras repetidas enumerados en el párrafo anterior a los 90 que ya teníamos, obtenemos un total de 100.

Veamos una función APL que calcula el número de variaciones con repetición de M elementos tomados de N en N:



```
[0]   Z←M VAR_R N
[1]   Z←M*N
```

Programa 10.

donde el asterisco es el símbolo APL que representa la elevación a una potencia. Veamos algunos ejemplos de su uso:

```
          10 VAR_R 2
100
          10 VAR_R 3
1000
          26 VAR_R 3
17576
```

Programa 11.

donde el último ejemplo nos calcula el número de palabras distintas de tres letras que podrían formarse con las 26 letras del alfabeto inglés, pudiendo repetirse las letras en cada palabra.



PERMUTACIONES

Las permutaciones son un caso particular de las variaciones, pues son todas las formas en que puede ordenarse un conjunto de M elementos y, por tanto, equivalen a las variaciones de M elementos tomadas de M en M. Esto significa que podemos utilizar la misma función VAR para calcular su número:

```
        6 VAR 6
720
        10 VAR 10
3628800
```

Programa 12.

Sin embargo, existe una forma más rápida y directa de calcularlas. En efecto, si nos fijamos en la fórmula general que nos da el número de variaciones sin repetición:

```
Mx(M-1)x(M-2)x...x(1+M-N)
```

Programa 13.

Observemos que, como en el caso de las permutaciones es N=M, se cumple que 1+M-N=1, y la fórmula anterior queda reducida a:

```
Mx(M-1)x(M-2)x...x2x1
```

Programa 14.

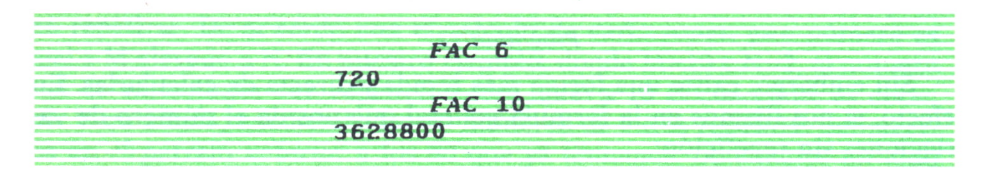
es decir, el número de permutaciones de los elementos de un conjunto de M elementos es igual al producto de todos los números enteros, desde 1 hasta M. Esta operación recibe un nombre en matemáticas: se llama «factorial» y se representa con un signo de admiración. En APL es posible calcular el factorial de un entero positivo cualquiera mediante una función recursiva como la siguiente:

```
[0]     Z←FAC M
[1]     Z←1
[2]     →(M=1)/0
[3]     Z←M×FAC M-1
```

Programa 15.

Veamos cómo funciona. La línea [1] asigna al resultado un valor inicial igual a 1. La línea [2] comprueba si el valor de M es igual a 1, y en tal caso, como el factorial de 1 es 1, el resultado correcto ha sido calculado y la función es abandonada. (La línea [2] puede interpretarse como «saltar a la línea 0, es decir, terminar la función, si M=1»). Por último, la línea [3], a la que se llega si M es distinto de 1, calcula el factorial de M como el producto de M por el factorial de M-1. Es fácil comprobar que el factorial tiene, en efecto, esta propiedad.

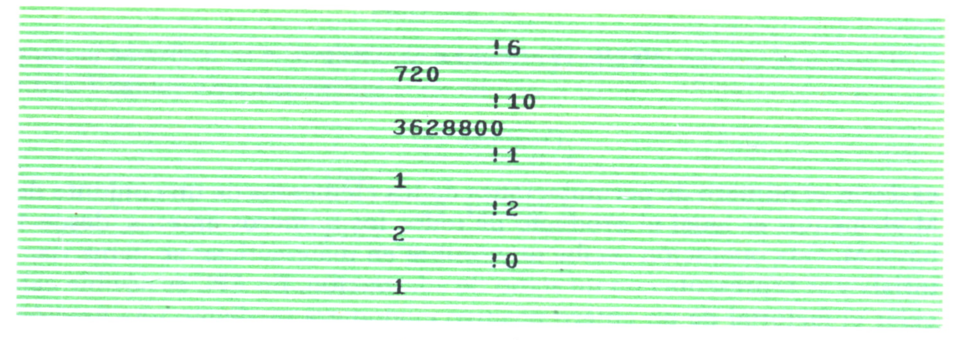
Veamos que al aplicar esta función a los ejemplos anteriores se obtienen los mismos resultados:



```
          FAC 6
720
          FAC 10
3628800
```

Programa 16.

Pero en APL hay otra forma mucho más sencilla de calcular el factorial, pues existe un símbolo que lo obtiene directamente. Al igual que en las matemáticas, el factorial se representa en APL con el signo de admiración pero, en vez de escribirlo a la derecha del número, se coloca a su izquierda. Veamos cómo se calcularían de esta forma los ejemplos anteriores y alguno más:



```
          !6
720
          !10
3628800
          !1
1
          !2
2
          !0
1
```

Programa 17.

Por convenio, se admite que !0 es igual a 1.

La función factorial tiene, en APL, las mismas propiedades que las funciones aritméticas ordinarias. Es decir, podemos aplicarla a series o tablas de datos directamente, sin necesidad de bucle alguno. Veamos algún ejemplo:

```
      !1 2 3 4 5 6 7 8 9 10
1 2 6 24 120 720 5040 40320 362880 3628800
      !⍳10
1 2 6 24 120 720 5040 40320 362880 3628800
```

Programa 18.

Podemos aplicar el concepto de permutación a la resolución de un problema sencillo: ¿De cuántas maneras distintas puede quedar ordenada una baraja de cartas al mezclarla? Es obvio que, como la baraja consta de 40 cartas, la solución a este problema será el número de permutaciones de un conjunto de 40 elementos, es decir, el factorial de 40. Veamos cuánto vale:

```
      !40
8.159152832E47
```

Programa 19.

Este número es tan grande, que tiene 48 cifras. Por tanto, rebasa la precisión de la máquina para números enteros y tiene que darse en notación exponencial. La primera parte del número (8.159152832) es la mantisa, y debe multiplicarse por 10 elevado a 47 para obtener el valor correcto. El valor aproximado del resultado es, por tanto, igual a 815915283200000000000000000000000000000000000000, un número inimaginablemente grande.

También podemos aplicar el concepto de factorial para calcular de otra manera el número de variaciones de M elementos tomados de N en N. Observemos que la fórmula que calcula el número de variaciones puede expresarse también así:

```
(!M)÷(!M-N)
```

Programa 20.

En efecto: !M es el producto de todos los números de 1 a M. !M-N es el producto de todos los números de 1 a M-N. Si dividimos el primer producto por el segundo, los números de 1 a M-N aparecerán tanto en el numerador como en el denominador, y podremos cancelarlos, con lo que el denominador desaparece y en el numerador quedará sólo el producto de los términos desde 1+M-N hasta M, que nos da el número de variaciones.

Aplicando esta propiedad, podremos construir la siguiente definición alternativa de la función VAR, que nos calcula el número de variaciones de M elementos tomados de N en N:

```
[0]    Z←M VAR N
[1]    Z←(!M)÷(!M-N)
```

Programa 21.

Es fácil comprobar que los resultados que se obtienen con esta función son idénticos a los que se obtenían con la versión que vimos al principio de este capítulo.



PERMUTACIONES CON REPETICION

Supongamos que tenemos un conjunto de M elementos, donde A de ellos son iguales y los demás son todos distintos entre sí. El cálculo de variaciones nos demuestra que el número de permutaciones es, en este caso, igual a !M dividido por !A.

Supongamos ahora que tenemos un conjunto de M elementos, donde éstos están repetidos y pueden clasificarse en grupos de elementos iguales entre sí. Los distintos grupos tendrán, respectivamente, A, B, ..., L elementos. En este caso, el número de permutaciones será igual a !M dividido por el producto de !A por !B por ... por !L.

Veamos una función APL que calcula el número de las permutaciones cuando hay elementos repetidos en el conjunto de partida:

```
[0]    Z←M PER_R G
[1]    Z←(!M)÷(×/!G)
```

Programa 22.

donde G es una serie de valores que nos da el número de elementos iguales que hay en cada grupo (no es necesario incluir los grupos que sólo tie-

nen un elemento). En efecto, !G nos calcula el factorial de cada uno de estos números y

```
(x/!G)
```

Programa 23.

nos da el producto de todos estos factoriales. Dividiendo por él !M, obtenemos el número de permutaciones deseado.

Veamos un ejemplo: sea el conjunto AAAABBCDE, que tiene 9 elementos. Si clasificamos éstos en grupos, según que sean iguales o no, nos salen los cinco grupos siguientes: AAAA, BB, C, D, E. El número de elementos de cada grupo es 4, 2, 1, 1, 1. Por tanto, el número de permutaciones posibles es (!9) dividido por el producto de !4 por !2 por !1 por !1 por !1. Como !1=1, podemos prescindir de los grupos que sólo tienen un elemento. Aplicando la función PER_R obtendremos directamente el número de permutaciones:

```
      9 PER_R 4 2 1 1 1
7560
      9 PER_R 4 2
7560
```

Programa 24.

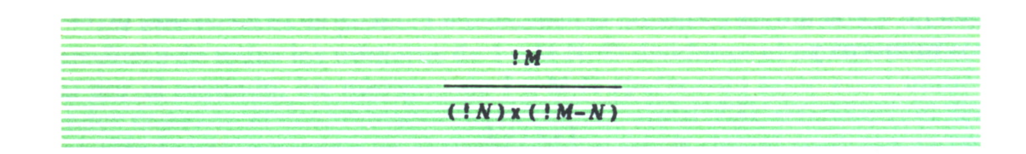


COMBINACIONES

Se llaman combinaciones de M objetos tomados de N en N al número de grupos distintos de N objetos que podemos formar con los M objetos de partida, sin importarnos el orden de los objetos dentro de cada grupo.

Con la terminología de teoría de conjuntos, diremos que las combinaciones de un conjunto de M objetos tomados de N en N es el número de subconjuntos distintos de dicho conjunto que tienen N elementos (o cuyo cardinal sea N).

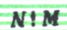
La combinatoria nos demuestra que dicho número, que llamaremos $C_{M,N}$, es igual a la siguiente expresión:



```
       !M
─────────────
(!N)x(!M-N)
```

Programa 25.

En APL sería trivial calcular este número, pero ni siquiera es necesario, pues existe una función primitiva (un símbolo del lenguaje) que nos lo calcula directamente. Ocurre que el número de combinaciones de M elementos tomados de N en N puede representarse en APL así:



```
N!M
```

Programa 26.

Veamos algunos ejemplos:

```
      2!5
10
      3!5
10
      4!5
5
      0!5
1
```

Programa 27.

Por convenio, se admite que $C_{M,\ 0}$, es igual a 1, cualquiera que sea M (mayor o igual que cero).

Por tratarse de una función básica del lenguaje APL, y al igual que la función factorial, la función de cálculo de los números combinatorios se puede aplicar a series o tablas de valores:

```
      0 1 2 3 4 5!5
1 5 10 10 5 1
```

Programa 28.

La expresión N!M se llama «número combinatorio».



TRIANGULO DE TARTAGLIA

Supongamos que hacemos M=1 y calculamos todos los números combinatorios posibles N!M. Como N debe ser entero positivo o cero y menor que M, sólo existen dos números combinatorios para M=1, a saber: 0!1 y 1!1, es decir, 1 y 1.

Supongamos ahora que M es igual a 2. Entonces existen tres números combinatorios posibles: 0!2, 1!2 y 2!2, es decir, 1, 2 y 1. De igual manera, para M=3 hay 4 números combinatorios, para M=4 hay cinco, y así sucesivamente. Si escribimos todos estos números sucesivamente, poniendo en la misma fila todos los que se obtienen para un valor determinado de M, y vamos aumentando el valor de M progresivamente al pasar de una fila a la siguiente, obtendremos un triángulo de valores como el siguiente:

```
M=1    1 1
M=2    1 2 1
M=3    1 3 3 1
M=4    1 4 6 4 1
M=5    1 5 10 10 5 1
M=6    1 6 15 20 15 6 1
```

Programa 29.

Esta disposición triangular de números se denomina «triángulo de Tartaglia». Los números que lo forman tienen algunas propiedades interesantes, como, por ejemplo, la siguiente:

«Un número del triángulo de Tartaglia es igual a la suma del que tiene encima y el que está a la izquierda de éste». (Compruébese.)

Vamos a escribir un programa APL que nos genere un triángulo de Tartaglia de N filas:

```
[0]     TARTAGLIA N;I
[1]     I←1
[2]   L:(0,⍳I)!I
[3]     I←I+1
[4]     →(I≤N)/L
```

Programa 30.

Veamos cómo funciona. En primer lugar, necesitamos un contador, que declararemos en la línea [0] como variable local, y que llamaremos I.

Inicialmente (en la línea [1]) le damos el valor 1 al contador I.

La línea [2] tiene una etiqueta (L) y genera la línea número I del triángulo de Tartaglia. Todos los números combinatorios de la misma fila se generan de golpe, utilizando la función representada por la letra griega «iota».

La línea [3] aumenta el valor del contador.

Finalmente, la línea [4] comprueba si el contador todavía no ha llegado al valor máximo deseado (el número de filas contenido en la variable N, argumento de la función TARTAGLIA). Si es así, transfiere el control a la línea cuya etiqueta es L (la línea [2]) y sigue calculando líneas del triángulo. En caso contrario, da por terminada la ejecución del programa. Esta línea puede leerse así: «Ir a la línea de etiqueta L si I es menor o igual que N.»

Veamos un ejemplo de la utilización de esta función:

```
      TARTAGLIA 10
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

Programa 31.

Otra propiedad curiosa de los números combinatorios es que la suma de todos los números que aparecen en una fila del triángulo de Tartaglia es igual a 2 elevado al número de la fila. Para comprobarlo, escribiremos una función, casi idéntica a la anterior, que en lugar de escribir el triángulo de Tartaglia escribe la suma de los elementos de sus filas:

```
[0]   TARTAGLIA1 N;I
[1]   I←1
[2]  L:+/(0,⍳I)!I
[3]   I←I+1
[4]   →(I≤N)/L
```

Programa 32.

Como se verá, la función es idéntica, excepto en la línea [2], donde se aplica la expresión +/ al conjunto de términos de la fila correspondiente del triángulo de Tartaglia. En APL, +/S (donde S es una serie de datos) significa la suma de todos los términos de S. Esta expresión equivale, por tanto, a la expresión ΣS, que se utiliza corrientemente en Matemáticas, pero tiene la ventaja de que puede generalizarse para cualquier otra operación aritmética. Por ejemplo, x/S representa en APL el producto de todos los elementos de S.

Ejecutemos la función TARTAGLIA1:

```
          TARTAGLIA1 10
    2
    4
    8
    16
    32
    64
    128
    256
    512
    1024
```

Programa 33.



EL BINOMIO DE NEWTON

Una de las aplicaciones de los números combinatorios es el cálculo de los coeficientes del desarrollo de la potencia de un binomio, $(A+B)^N$. Puede demostrarse con facilidad que $(A+B)^N = (0!N)\times A^N \times B^0 + (1!N)\times A^{N-1} \times B^1 + (2!N)\times A^{N-2}\times B^2 + \ldots + ((N-2)!N)\times A^2\times B^{N-2} + ((N-1)!N)\times A^1\times B^{N-1} + (N!N)\times A^0\times B^N$ es decir, que los coeficientes del desarrollo del binomio son los números que aparecen en la fila número N del triángulo de Tartaglia.

Vamos a hacer una función que calcule el binomio de Newton por este método:

```
[0]    Z←B BINOMIO N;X;Y;COEF;C;POTX;POTY
[1]    X←B[1]
[2]    Y←B[2]
```

```
[3]   C←0,⍳N
[4]   COEF←C!N
[5]   POTX←X*N-C
[6]   POTY←Y*C
[7]   Z←+/COEF×POTX×POTY
```

Programa 34.

Veamos cómo funciona. El argumento B debe ser una serie con los dos términos del binomio. El argumento N será la potencia. En la línea [0] definimos también las variables locales a este programa que vamos a utilizar: X, Y, COEF, C, POTX y POTY.

En la línea [1] obtenemos X (primer término del binomio) como primer elemento de B.

En la línea [2] obtenemos Y (segundo término del binomio) como segundo elemento de B.

En la línea [3] guardamos en C el conjunto de los números desde cero hasta N. Estos serán los términos izquierdos de los números combinatorios y los exponentes de los dos términos del binomio.

La línea [4] calcula la línea N del triángulo de Tartaglia y la guarda en la variable COEF. Ya tenemos los coeficientes del desarrollo del binomio.

La línea [5] calcula las potencias de X (primer término del binomio) en orden descendente (desde N hasta 0). Recuérdese que el asterisco (*) representa en APL la elevación a una potencia.

La línea [6] calcula las potencias de Y (segundo término del binomio) en orden ascendente.

Por último, la línea [7] calcula la suma de los productos de la serie de los coeficientes por las dos series de potencias. Este es el resultado de la aplicación del binomio de Newton.

Veamos un ejemplo:

```
          3 5 BINOMIO 3
512
```

Programa 35.

Ahora vamos a comprobar que este valor es correcto. Calculemos directamente la potencia del binomio, sin desarrollarla:

```
          (3+5)*3
512
```

Programa 36.

Como puede verse, el resultado es el mismo.

POLINOMIOS 4



N polinomio de una sola variable es una suma de términos, cada uno de los cuales consta de un coeficiente constante que multiplica a una potencia de grado entero de una variable. Se denomina grado del polinomio al grado o exponente más elevado entre todos los términos. Por tanto, la forma más general posible de un polinomio de grado «n» en la variable «x» será:

$$a_0 + a_1x + ... + a_{n-1}x^{n-1} + a_nx^n$$

donde algunos de los coeficientes (excepto el último, a_n pueden ser iguales a cero.

Un polinomio determinado queda totalmente definido por la lista ordenada de sus coeficientes, pues el número de éstos indica el grado del polinomio. Supondremos en lo sucesivo que los coeficientes vienen siempre ordenados en el orden ascendente de los grados de los términos sucesivos, es decir, que el polinomio anterior quedaría definido por la siguiente sucesión:

$$a_0\ a_1\ ...\ a_{n-1}\ a_n$$

Veamos un ejemplo: la sucesión de coeficientes

5 2 0 3

representará al polinomio

$$5 + 2x + 3x^3$$

ya que el término en x^2 falta, por ser su coeficiente igual a cero.

Una vez tenemos definido un polinomio por la sucesión ordenada de sus coeficientes, podemos calcular el valor que toma el polinomio para cada valor de x determinado. Por ejemplo, el polinomio anterior toma el valor 5 para x=0 (ya que $5 = 5 + 2 \cdot 0 + 3 \cdot 0^3$), el valor 10 para x=1 (pues $10 = 5 + 2 \cdot 1 + 3 \cdot 1^3$) y el valor 0 para x=-1 (porque $0 = 5 + 2(-1) + 3(-1)^3$).

Vamos a construir un programa APL que calcule el valor que toma un polinomio para cierto valor de su variable x, si le damos la lista de los coeficientes y el valor de x.

```
[0]  Z←C POLI X
[1]   Z←+/C×X*(¯1+⍳⍴C)
```

Programa 1.

Veamos cómo funciona. Además de la lista ordenada de los coeficientes (C) y del valor de la variable independiente (X) necesitamos también la lista ordenada de los exponentes de X. Esta lista es muy fácil de calcular. En efecto: supongamos que C tiene 4 coeficientes. En tal caso, los exponentes serán siempre los números 0,1,2,3. Si C tuviera 6 coeficientes, los exponentes serían 0,1,2,3,4,5, y así sucesivamente. Podemos ver que, si C tiene *n* elementos, los exponentes serán los números comprendidos entre *0* y *n–1*, en ese orden.

Pues bien: en APL, el número «n» de elementos que tiene una variable C es

```
⍴C
```

Programa 2.

y la lista de números comprendidos entre 1 y «n» será, por tanto:

```
⍳⍴C
```

Programa 3.

Como lo que queremos es la lista desde 0 hasta «n-1», para obtenerla bastará con restarle 1 a la lista anterior:

```
¯1+⍳⍴C
```

Programa 4.

Ya tenemos, por tanto, la lista de los exponentes. Ahora hay que elevar X a cada uno de esos exponentes. Nada más fácil: basta con escribir:

```
X*(¯1+⍳⍴C)
```

Programa 5.

Además, hay que multiplicar cada término por el coeficiente correspondiente. Como C tiene los coeficientes, en número igual al de los exponentes, bastará con multiplicar las dos series entre sí. Pues en efecto, en APL el producto de dos series de igual número de términos se realiza elemento a elemento, obteniéndose otra serie con el mismo número de elementos que las dos de partida.

```
CxX*(¯1+ιρC)
```

Programa 6.

Finalmente, ya sólo falta sumar todos los términos de la serie resultante (ya sabemos que esto se consigue mediante los símbolos «+/». Así obtenemos el valor del polinomio, tal y como se indica en la línea [1] del programa anterior.

Veamos algunos ejemplos:

```
      5 2 0 3 POLI 0
5
      5 2 0 3 POLI 1
10
      5 2 0 3 POLI ¯1
0
      5 2 0 3 POLI 2
33
      5 2 0 3 POLI 5
390
      3 1 4 POLI 5
108
```

Programa 7.

Obsérvese que esta función nos sirve para calcular los valores, no sólo del polinomio de nuestro ejemplo, sino también de cualquier otro. Basta con poner a la izquierda de POLI los coeficientes respectivos.



SUMA DE POLINOMIOS

Supongamos que tenemos dos polinomios distintos y queremos obtener su suma. Cada uno de los polinomios vendrá representado por el vector de sus coeficientes, y el resultado será el vector de los coeficientes del polinomio suma. El siguiente programa APL obtiene la suma de los dos polinomios:

```
[0]   Z←C1 POLSUM C2;N
[1]    N←(ρC1)⌈(ρC2)
[2]    Z←(N↑C1)+(N↑C2)
```

Programa 8.

Veamos con un ejemplo cómo lo realiza. Supongamos que C1 es igual a 3 1 4 y C2 vale 5 2 0 3 (los dos polinomios de nuestro ejemplo anterior). Como los dos polinomios tienen distinto grado (C1 tiene menos términos que C2), para sumarlos tenemos que igualar el grado de los dos, añadiendo al de menos términos los ceros necesarios hasta que iguale el grado del mayor. Es decir, tendremos que convertir C1 en la serie 3 1 4 0. Una vez conseguido esto, el polinomio suma tendrá como coeficientes la suma de los coeficientes respectivos de ambos polinomios.

La expresión

```
(ρC1)⌈(ρC2)
```

Programa 9.

calcula el máximo entre los dos términos, que no son otra cosa que el número de coeficientes de los dos polinomios. Por tanto, la línea [1] del programa asigna a la variable N el valor de dicho máximo. Este es el número de términos en que tenemos que igualar ambos polinomios antes de sumarlos.

La expresión

```
N↑V
```

Programa 10.

donde N es un número entero positivo y V es una serie de datos, genera una nueva serie compuesta por los N primeros términos de V. Si V tiene menos de N términos, le añade ceros al final hasta llegar al número N pedido. Por tanto, puede verse con facilidad que la línea [2] del programa **POLSUM** hace uso de esta operación para igualar la longitud de ambas series y sumarlas término a término.

Veamos algún ejemplo de aplicación:

```
        5 2 0 3 POLSUM 3 1 4
8 3 4 3
        8 3 4 3 POLI 5
498
        (5 2 0 3 POLI 5)+(3 1 4 POLI 5)
498
        (5 2 0 3 POLSUM 3 1 4) POLI 5
498
```

Programa 11.

Los ejemplos muestran que esta función puede combinarse con la función **POLI** para calcular el valor del polinomio suma de otros dos para un valor determinado de su variable independiente. Demuestran también que lo que se obtiene es, en efecto, la suma de los valores de ambos polinomios para el mismo valor de la variable independiente.



PRODUCTO DE POLINOMIOS

De igual manera que en el caso de la suma, la función siguiente calcula el producto de dos polinomios, definidos por sus coeficientes respectivos:

```
[0]   Z←C1 POLPROD C2
[1]    Z←+⌿(1-⍳⍴C1)⌽C1∘.×C2,(¯1+⍴C1)⍴0
```

Programa 12.

Para ver cómo funciona vamos a seguir un ejemplo:

```
        5 2 0 3 POLPROD 3 1 4
 15 11 22 17 3 12
```

Programa 13.

Mientras la función POLPROD se está ejecutando, la variable C1 será igual al polinomio definido por los coeficientes 5 2 0 3, y la función C2 al polinomio 3 1 4:

```
      C1
5 2 0 3
      C2
3 1 4
```

Programa 14.

La expresión

```
      (¯1+⍴C1)⍴0
0 0 0
```

Programa 15.

genera tantos ceros como elementos tiene C1 menos 1. La expresión

```
      C2,(¯1+⍴C1)⍴0
3 1 4 0 0 0
```

Programa 16.

concatena dicha serie de ceros a la derecha de los valores de C2. A continuación, la expresión

```
      C1∘.×C2,(¯1+⍴C1)⍴0
15  5 20  0  0  0
 6  2  8  0  0  0
 0  0  0  0  0  0
 9  3 12  0  0  0
```

Programa 17.

forma la tabla de multiplicar de los coeficientes de C1 por los coeficientes de C2 seguidos por los tres ceros. Obsérvese que el polinomio producto se obtendrá sumando por columnas dichos coeficientes, pero antes debemos rotar las filas. La primera fila está en su sitio, la segunda debe rotar una posición hacia la derecha, la tercera dos posiciones, y la tercera tres.

La expresión

```
      (1-⍳⍴C1)
0 ¯1 ¯2 ¯3
```

Programa 18.

genera precisamente el número de posiciones que habrá de rotar cada fila de la tabla de multiplicar anterior. Los números negativos indican que la rotación será hacia la derecha. Los números positivos, en su caso, indicarían rotaciones hacia la izquierda. Finalmente, el símbolo representado por la letra griega «fi» realiza las rotaciones correspondientes:

```
      (1-⍳⍴C1)⌽C1∘.×C2,(¯1+⍴C1)⍴0
15  5 20  0  0  0
 0  6  2  8  0  0
 0  0  0  0  0  0
 0  0  0  9  3 12
```

Programa 19.

Por último, el signo + junto a la barra cruzada realiza la suma por columnas de la tabla obtenida, calculando así los coeficientes del producto de los dos polinomios:

```
      +⌿(1-⍳⍴C1)⌽C1∘.×C2,(¯1+⍴C1)⍴0
15 11 22 17 3 12
```

Programa 20.



RAICES DE UN POLINOMIO

Se llaman «raíces» de un polinomio a los valores de X que hacen el valor del polinomio igual a cero. La función **POLI**, que nos calcula el valor del polinomio para un valor de X determinado nos puede ayudar a descubrir las raíces del polinomio.

Sea, por ejemplo, el polinomio

$$4 - 8x + 3x^2$$

definido por los coeficientes 4 –8 3. Para descubrir sus raíces podríamos emplear el método general de resolución de ecuaciones de segundo grado, pero vamos a hacerlo de una manera diferente, que también puede aplicarse a ecuaciones (o polinomios) de cualquier grado. Este método consiste en buscar cambios de signo en los valores del polinomio. Como todos los polinomios son funciones continuas, los ceros (o raíces) tienen que estar situados en algún valor de X intermedio entre los que dan lugar a valores de signos opuestos.

Calculemos los valores del polinomio anterior para distintos valores de X:

```
      4 ¯8 3 POLI ¯1
15
      4 ¯8 3 POLI 0
4
      4 ¯8 3 POLI 1
¯1
      4 ¯8 3 POLI 2
0
      4 ¯8 3 POLI 3
7
```

Programa 21.

Con los resultados obtenidos, conocemos ya una de las raíces del polinomio: X=2. Además, sabemos que la otra raíz tiene que estar comprendida entre X=0 y X=1, pues ahí se produce un cambio de signo en los valores del polinomio. Para buscar el valor correcto, bastará con que vayamos tanteando con valores de X intermedios entre 0 y 1. Probaremos, en primer lugar, con X=0.5:

```
      4 ¯8 3 POLI 0.5
0.75
```

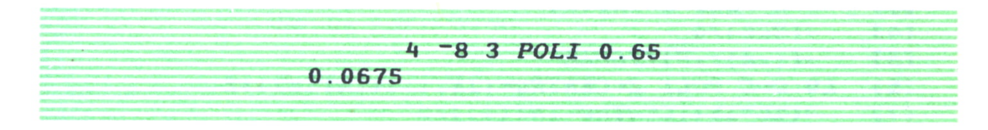
Programa 22.

El resultado obtenido (0.75) es positivo, por lo que el cambio de signo se produce ahora entre X=0.5 y X=1. La raíz tiene que estar en algún punto intermedio. Probemos con X=0.75:

```
      4 ¯8 3 POLI 0.75
¯0.3125
```

Programa 23.

Ahora nos sale un resultado negativo, lo que significa que el cambio de signo estará situado entre X=0.5 y X=0.75. Probaremos, por tanto, con X=0.65:



```
      4 ¯8 3 POLI 0.65
0.0675
```

Programa 24.

Otra vez nos sale positivo. Por tanto, el cambio de signo está entre X=0.65 y X=0.75. Probemos con X=0.67:

```
      4 ¯8 3 POLI 0.67
¯0.0133
```

Programa 25.

Se observará que cada vez estamos más cerca de obtener el valor cero. Ahora nos ha salido un resultado negativo, lo que quiere decir que la raíz (el cambio de signo) debe estar entre X=0.65 y X=0.67. Probaremos, pues, con X=0.66:

```
      4 ¯8 3 POLI 0.66
0.0268
```

Programa 26.

Este nuevo resultado positivo nos acota ahora la posición de la raíz entre X=0.66 y X=0.67. Parece evidente que nos estamos acercando cada vez más al valor 2/3. Probémoslo:

```
      4 ¯8 3 POLI 2÷3
0
```

Programa 27.

Luego, efectivamente, 2/3 es la otra raíz del polinomio.

El método que acabamos de ver es susceptible de ser llevado a cabo por un ordenador. Vamos a construir un programa APL que lo realice automáticamente.

```
[0]   Z←INT RAIZ C;X;A;B;D;CENTRO
[1]   L:A←C POLI Z←INT[1]
[2]    →(A=0)/0
[3]    B←C POLI Z←INT[2]
[4]    →(B=0)/0
[5]    →((×A)=×B)/IGUAL
[6]   L0:D←C POLI Z←CENTRO←(INT[1]+INT[2])÷2
[7]    →(1E¯15<|CENTRO-INT[1])/0
[8]    →((×D)=×A)/L1
[9]    INT[2]←CENTRO
[10]   →L0
[11]  L1:INT[1]←CENTRO
[12]   →L0
[13]  IGUAL:CENTRO←INT[1]+0.01×(⍳100)×(INT[2]-INT[1])
[14]   D←C POLI1 CENTRO
[15]   D←((×D)≠×A)⍳1
[16]   Z←'NO LA ENCUENTRO'
[17]   →(D=101)/0
[18]   Z←(CENTRO[D],INT[2])RAIZ C
```

Programa 28.

La función RAIZ tiene dos argumentos. El izquierdo es el intervalo donde deseamos que busque una raíz del polinomio. El derecho es el conjunto de coeficientes del polinomio, dados como siempre. Se observará que este programa hace uso de la función POLI, que hemos visto más arriba, para calcular el valor del polinomio para un valor de X determinado. También utiliza la función POLI1, que es una versión de POLI que permite calcular de una sola vez los valores del polinomio para varios valores de X, dados en forma de vector. Veamos la función POLI1, junto con un par de ejemplos de su utilización:

```
      [0]  Z←C POLI1 X
      [1]   Z←+/(((⍴X),⍴C)⍴C)×X∘.*(¯1+⍳⍴C)
            4 ¯8 3 POLI1 ¯1 0 1 2 3
      15 4 ¯1 0 7
            4 ¯8 3 POLI1 0.5 0.75 0.65 0.67 0.66
      0.75 ¯0.3125 0.0675 ¯0.0133 0.0268
```

Programa 29.

Veamos un ejemplo del uso de la función RAIZ:

```
        ¯5 5 RAIZ 4 ¯8 3
2
        2.00001 5 RAIZ 4 ¯8 3
NO LA ENCUENTRO
        ¯5 1.99999 RAIZ 4 ¯8 3
0.6666666667
```

Programa 30.

Vemos que, al darle el intervalo [–5,5], el programa RAIZ ha sido capaz de descubrir la raíz X=2. Para buscar la otra, tenemos que excluir la raíz 2, por lo que probamos primero con el intervalo (2,5]. Para hacerlo abierto por la izquierda (es decir, para que no vuelva a encontrar X=2) damos como origen del intervalo el valor 2.00001. En este caso, el programa no logra encontrar ninguna raíz. Probamos entonces con el intervalo [-5,2), abierto por la derecha (para lo que damos como extremo del intervalo el valor 1.99999). Ahora sí encuentra el programa la segunda raíz, que sale igual a 2/3.



DERIVADA DE UN POLINOMIO

La derivada de un polinomio es otro polinomio de grado inferior en una unidad, cuyos coeficientes se obtienen multiplicando los coeficientes del polinomio por los exponentes de los términos respectivos, y excluyendo el primero de los coeficientes obtenidos, que siempre sale igual a cero.

Definida así, es muy fácil construir una función APL que calcule los coeficientes del polinomio derivada, partiendo de los coeficientes de un polinomio cualquiera:

```
[0]  Z←POLIDER C
[1]  Z←1↓C×¯1+⍳⍴C
```

Programa 31.

Veamos algunos ejemplos:

```
        POLIDER 4 ¯8 3
¯8 6
        POLIDER 5 2 0 3
2 0 9
        POLIDER POLIDER 5 2 0 3
0 18
        POLIDER 3 1 4
1 8
```

Programa 32.

Se observará que la derivada segunda se obtiene, simplemente, aplicando dos veces la función POLIDER. Esta función puede combinarse con RAIZ para calcular la posición de los máximos y mínimos de un polinomio. En efecto, se sabe que estos puntos especiales coinciden siempre con las raíces de la derivada del polinomio. Por tanto, podremos hallarlos derivando primero el polinomio y después buscando las raíces del polinomio resultante, utilizando para ello la función RAIZ. De igual manera, los puntos de inflexión podrán hallarse calculando las raíces de la derivada segunda del polinomio. Veamos algunos ejemplos:

```
        ¯5 5 RAIZ POLIDER 4 ¯8 3
1.333333333
        ¯5 5 RAIZ POLIDER POLIDER 4 ¯8 3
NO LA ENCUENTRO
        ¯5 5 RAIZ POLIDER 5 2 0 3
NO LA ENCUENTRO
        ¯5 5 RAIZ POLIDER POLIDER 5 2 0 3
¯5.551115123E¯16
```

Programa 33.

Es decir, el polinomio $4\text{-}8X+3X^2$ tiene el mínimo en el punto X=4/3, pero no tiene ningún punto de inflexión en el intervalo [-5,5] (ni fuera de él, pues los polinomios de segundo grado no tienen puntos de inflexión). En cambio, el polinomio $5+2X+3X^3$ no tiene máximos ni mínimos en el intervalo [-5,5], pero tiene un punto de inflexión para X=0.

VECTORES, MATRICES Y SISTEMAS DE ECUACIONES 5



VECTORES

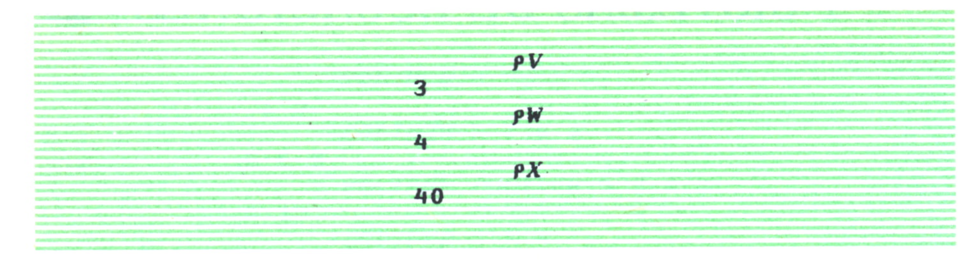


COMO hemos visto, una serie de números puede considerarse como un conjunto, si el orden de sus elementos no tiene importancia, pero también puede considerarse como un vector, aunque en este caso sí importa dicho orden. En APL los vectores se generan exactamente igual que los conjuntos, asignando una serie de datos al nombre del vector:

```
      V←1 3 5
      V
1 3 5
      W←2 4 6 8
      W
2 4 6 8
      X←⍳40
      X
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
      21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
      36 37 38 39 40
```

Programa 1

Se dice que un vector está definido en un espacio de «n» dimensiones, si el número de sus elementos es igual a «n». Por tanto, el número de dimensiones de un vector se podrá obtener de la misma manera que el cardinal de un conjunto: con la operación «rho». En los ejemplos anteriores, V, W y X son vectores de tres, cuatro y cuarenta dimensiones, respectivamente.



```
            ρV
      3
            ρW
      4
            ρX
      40
```

Programa 2.

En los apartados sucesivos vamos a ver algunas operaciones que pueden realizarse con los vectores:



Suma de los elementos de un vector

La suma de los elementos de un vector se obtiene anteponiendo al nombre del vector los símbolos +/. Por ejemplo:

```
            +/V
      9
            +/W
      20
            +/X
      820
```

Programa 3.

Obsérvese que esta operación se realiza correctamente cualquiera que sea el número de los elementos del vector (su dimensión).

La suma progresiva de los elementos de un vector se obtiene anteponiendo al nombre del vector los símbolos +/. Por ejemplo:

```
       +\V
 1 4 9
       +\W
 2 6 12 20
       +\X
 1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136
       153 171 190 210 231 253 276 300 325 351 378
       406 435 465 496 528 561 595 630 666 703 741
       780 820
```

Programa 4.

Obsérvese que el primer elemento de la suma progresiva es siempre igual al primer elemento del vector, mientras que el último elemento de la suma progresiva es igual a la suma de los elementos del vector.

Obsérvese también que la suma progresiva de los N primeros números consecutivos (como en +/×) es igual a los N primeros números triangulares.

Producto de los elementos de un vector

De igual manera que la suma, el producto de los elementos de un vector se obtiene anteponiendo al nombre del vector los símbolos x/. En cuanto al producto progresivo, se obtiene con los símbolos ×/. Por ejemplo:

```
          ×/V
15
          ×/W
384
          ×/X
8.159152832E47
          ×\V
1 3 15
          ×\W
2 8 48 384
```

Programa 5.

Obsérvese que el primer elemento del producto progresivo es siempre igual al primer elemento del vector, mientras que el último elemento del producto progresivo es igual al producto de los elementos del vector.

Elementos máximo y mínimo de un vector

Los elementos máximo y mínimo de un vector se obtienen de forma equivalente a la suma y el producto de sus elementos, utilizando en lugar de los símbolos de la suma o la multiplicación los símbolos APL de las operaciones máximo y mínimo, respectivamente:

```
          ⌈/V
5
          ⌈/W
8
```

```
          ⌈/X
40
          ⌈/5 3 ¯2 12.5 ¯23
12.5
          ⌊/V
1
          ⌊/W
2
          ⌊/X
1
          ⌊/5 3 ¯2 12.5 ¯23
¯23
```

Programa 6.



Inversión y rotación de los elementos de un vector

Para invertir el orden de los elementos de un vector basta con aplicar la siguiente operación APL:

```
      ⌽V
5 3 1
      ⌽W
8 6 4 2
      ⌽X
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
      23 22 21 20 19 18 17 16 15 14 13 12 11 10 9
      8 7 6 5 4 3 2 1
```

Programa 7.

Para rotar los elementos en forma circular (los que se pierden por un lado vuelven a entrar por el otro) se utiliza la misma operación APL, pero colocando a su izquierda un número entero que indica el número de posiciones que deben desplazarse los elementos. Si este número es positivo, el desplazamiento o rotación tiene lugar hacia la izquierda. Si es negativo, hacia la derecha.

```
        2ΦV
5 1 3
        ¯2ΦV
3 5 1
        1ΦW
4 6 8 2
        ¯1ΦW
8 2 4 6
```

Programa 8.



Suma o producto por un escalar

Para sumar, restar o multiplicar un vector y un escalar (un número) basta utilizar el signo de la suma, de la resta o de la multiplicación, como si se tratara de números ordinarios. Por ejemplo:

```
        3+V
4 6 8
        W×2
4 8 12 16
```

Programa 9.



Suma o producto de dos vectores

Para sumar, restar o multiplicar elemento a elemento dos vectores del mismo número de elementos, basta con separar sus nombres por el signo de la suma, de la resta o de la multiplicación, respectivamente, como si se tratara de números ordinarios. Por ejemplo:

```
        V1←2 4 6 8
        V2←1 3 5 7
        V1+V2
3 7 11 15
        V1×V2
2 12 30 56
        V1×V1
4 16 36 64
```

Programa 10.



Módulo de un vector

Se llama módulo de un vector a la raíz cuadrada de la suma de los cuadrados de sus elementos. La siguiente función APL calcula el módulo de cualquier vector:

```
[0]  Z←MODULO V
[1]   Z←(+/V×V)*0.5
```

Programa 11.

Para comprenderla, obsérvese que V × V es el vector cuyos elementos son los cuadrados de los elementos de V, por lo que +/V × V será la suma de los cuadrados de los elementos de V. La elevación a la potencia 0.5 es idéntica a la raíz cuadrada, como se sabe por matemáticas elementales.

Veamos algunos ejemplos del cálculo del módulo de un vector:

```
      MODULO V
5.916079783
      MODULO W
10.95445115
      MODULO X
148.7951612
```

Programa 12.



Producto escalar de dos vectores

El producto escalar de dos vectores del mismo número de dimensiones es igual a la suma de los productos de sus componentes. Si llamamos a los dos vectores V y W, y a su elemento i-ésimo lo representamos con los símbolos V_i y W_i, respectivamente, el producto escalar de V y W será igual a:

$$\sum_{i} V_i \times W_i$$

La siguiente función APL calcula el producto escalar de dos vectores V y W:

```
[0]  Z←V PROD_ESC W
[1]   Z←+/V×W
```

Programa 13.

La forma de calcularlo corresponde exactamente a la definición anterior. Veamos algunos ejemplos de su funcionamiento:

```
      V1 PROD_ESC V2
100
      1 2 3 PROD_ESC 3 2 1
10
```

Programa 14.

También puede expresarse el producto escalar de dos vectores con la operación representada por los símbolos «+.x», como en los siguientes ejemplos:

```
      V1+.xV2
100
      1 2 3 +.x 3 2 1
10
```

Programa 15.

La ventaja de esta notación es que, en lugar de los símbolos + y x, puede utilizarse cualquier par de funciones aritméticas APL. La segunda operación se aplicará elemento a elemento entre los dos vectores, y a continuación se aplica la primera operación en forma de reducción entre todos los resultados obtenidos. Por ejemplo, podemos comprobar si dos vectores son iguales de la siguiente manera:

```
      V1∧.=V2
0
      V1∧.=V1
1
      V1∧.=W
1
```

Programa 16.



MATRICES

Una tabla de números puede considerarse como una matriz. Muchas de las operaciones con matrices son triviales en APL, como veremos a continuación.

Para generar una matriz se puede utilizar el símbolo «rho», colocando a su izquierda el número de filas y de columnas que va a tener la matriz y a su derecha los valores de la matriz, expresados por filas:

```
          M←3 4ρι12
          M
  1  2  3  4
  5  6  7  8
  9 10 11 12
          M1←3 3ρ2 4 6 1 3 5 1 3 3
          M1
2 4 6
1 3 5
1 3 3
          M2←3 3ρι9
          M2
1 2 3
4 5 6
7 8 9
```

Programa 17.

En los apartados sucesivos vamos a ver algunas operaciones que pueden realizarse con matrices:



Suma o producto por un escalar

Para sumar, restar o multiplicar una matriz y un escalar (un número) basta utilizar el signo de la suma, de la resta o de la multiplicación, respectivamente, como si se tratara de números ordinarios. Por ejemplo:

```
      3+M
 4  5  6  7
 8  9 10 11
12 13 14 15
      M1×2
 4  8 12
 2  6 10
 2  6  6
```

Programa 18.



Suma o producto de dos matrices

Para sumar, restar o multiplicar elemento a elemento dos matrices del mismo número de filas y de columnas basta con separar sus nombres por el signo de la suma, de la resta o de la multiplicación, respectivamente, como si se tratara de números ordinarios. Por ejemplo:

```
      M1+M2
 3  6  9
 5  8 11
 8 11 12
      M2-M1
¯1 ¯2 ¯3
 3  2  1
 6  5  6
      M1×M2
 2  8 18
 4 15 30
 7 24 27
```

Programa 19.



Transposición de una matriz

Se llama matriz transpuesta de una dada a la que se obtiene cambiando filas por columnas y columnas por filas. Dicho de otro modo, el elemento a'_{ij} de la matriz transpuesta es igual al elemento a_{ji} de la matriz original.

Veamos cómo se obtiene en APL la matriz transpuesta de una dada:

```
      ⍉M
 1  5  9
 2  6 10
 3  7 11
 4  8 12
      ⍉M1
2 1 1
4 3 3
6 5 3
      ⍉M2
1 4 7
2 5 8
3 6 9
```

Programa 20.

Producto matricial

Dadas dos matrices, M1 y M2, de elementos a_{ij} y b_{ij}, respectivamente, tales que el número de columnas de la primera es igual al número de filas de la segunda, se llama producto matricial de ambas a otra matriz, M3, que tiene tantas filas como filas tiene M1 y tantas columnas como columnas tiene M2, y cuyos elementos c_{ij} se obtienen así:

$$c_{ij} = \sum_{i} a_{ik} \times b_{kj}$$

En APL se representa el producto matricial de dos matrices con los mismos símbolos que el producto escalar de vectores, puesto que ambas operaciones son equivalentes si consideramos al primer vector como una matriz de una sola fila y al segundo como una matriz de una sola columna. Veamos algunos ejemplos:

```
      M1+.×M
 76   88 100 112
 61   70  79  88
 43   50  57  64
      M1+.×M2
60 72 84
48 57 66
34 41 48
      M2+.×M2
 30  36  42
 66  81  96
102 126 150
```

Programa 21.

También puede multiplicarse una matriz por un vector que tenga tantos elementos como columnas la matriz, o bien un vector por una matriz que tenga tantas filas como elementos el vector. El resultado, en ambos casos, será un vector con tantos elementos como la otra dimensión de la matriz.

```
      V
1 3 5
      V+.×M
61 70 79 88
      M+.×W
60 140 220
```

Programa 22.

Matriz inversa

Se llama traza o diagonal principal de una matriz cuadrada M de elementos m_{ij} al conjunto de elementos m_{ii} cuyo número de fila coincide con su número de columna.

En APL la traza de una matriz se obtiene así:

```
        1 1⍉M1
2 3 3
        1 1⍉M2
1 5 9
        1 1⍉(M1+.×M2)
60 57 48
```

Programa 23.

Se llama matriz unidad a toda matriz cuadrada cuya traza está compuesta únicamente de unos, mientras que los restantes elementos son todos iguales a cero.

Veamos una función APL que construye una matriz unidad de N filas y N columnas, cualquiera que sea N, junto con algunos ejemplos de su uso:

```
[0]   Z←M_UNIDAD N
[1]     Z←(N,N)⍴1,N⍴0
        M_UNIDAD 2
1 0
0 1
        M_UNIDAD 3
1 0 0
0 1 0
0 0 1
        M_UNIDAD 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

Programa 24.

La función anterior se apoya en el hecho de que, cuando se dan menos valores de los necesarios para la creación de una matriz por medio de la operación «rho», los valores dados vuelven a utilizarse cíclicamente.

Se llama matriz inversa de una dada a aquella matriz que, multiplicada matricialmente por ella, da como resultado la matriz unidad.

En APL existe un símbolo que nos permite obtener directamente la matriz inversa de una matriz dada. Este símbolo se parece a la ficha «pito doble» del dominó, y procede de encerrar el símbolo de dividir en un cuadrado que representa la matriz.

```
      MM←3 3⍴3 2 3 2 3 1 3 1 2
      MM
3 2 3
2 3 1
3 1 2
      ⌹MM
¯0.625              0.125               0.875
 0.125              0.375              ¯0.375
 0.875             ¯0.375              ¯0.625
      MM+.×⌹MM
 1.000000000E0     ¯5.551115123E¯17     0.000000000E0
 2.220446049E¯16    1.000000000E0       2.220446049E¯16
 4.440892099E¯16    0.000000000E0       1.000000000E0
```

Programa 25.

Se observará que el producto de la matriz por su inversa no es exactamente igual a la matriz unidad, aunque se aproxima mucho. Los elementos de la diagonal principal o traza son, efectivamente, iguales a la unidad, pero algunos de los elementos restantes no son exactamente iguales a cero, aunque sí son muy pequeños. Esto se debe a que los números decimales no pueden guardarse exactamente en la memoria de un ordenador, sino que suelen tener un error muy pequeño (del orden de la decimoquinta cifra significativa) que a veces aparece visiblemente o se amplifica como resultado de ciertos cálculos.



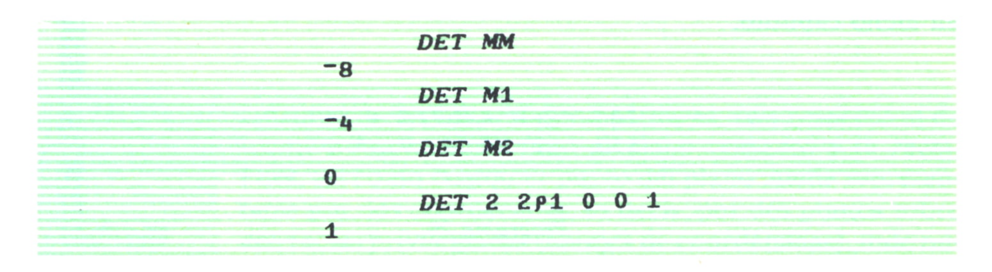
Determinante de una matriz

La siguiente función APL calcula el determinante de una matriz cuadrada. Se trata de una función recursiva, que calcula el determinante de una matriz N × N en función del determinante de otra matriz de orden inferior en una unidad (N–1) × (N–1).

```
[0]   Z←DET M;K
[1]     M[K,1;]←M[1,K←K⍳⌈/K←|M[;1];]
[2]     Z←(1E¯10<|M[1;1])×M[1;1]×¯1*K≠1
[3]     →(∨/(1=⍴M),(Z=0))/0
[4]     Z←Z×DET 1 1↓M-M[;1]∘.×M[1;]÷M[1;1]
```

Programa 26.

Veamos algunos ejemplos de su utilización:



```
              DET MM
 ¯8
              DET M1
 ¯4
              DET M2
 0
              DET 2 2⍴1 0 0 1
 1
```

Programa 27.



SISTEMAS DE ECUACIONES

Sea el sistema de ecuaciones

$$3X + 2Y + 3Z = 10$$
$$2X + 3Y + Z = 5$$
$$3X + Y + 2Z = 3$$

Para resolverlo, debemos construir la matriz de sus coeficientes (que en los ejemplos anteriores hemos llamado MM). Entonces, la solución del sistema puede obtenerse en APL de la siguiente forma:

```
              10 5 3 ⌹ MM
 ¯3 2 5
```

Programa 28.

Es decir, X = -3, Y = 2, Z = 5 es la solución del sistema.

El procedimiento es totalmente general, y no depende del número de ecuaciones, que en principio debe ser igual al número de incógnitas.



Ajuste polinómico por mínimos cuadrados

Sean V y W dos vectores del mismo número de elementos. Supongamos que V contiene los valores de una variable independiente y W los de una variable dependiente de la anterior. ¿Qué polinomio de grado N produce el mejor ajuste de los valores de W en función de los de V? Pues bien: la siguiente función APL realiza dicho ajuste y obtiene los coeficientes del polinomio en el mismo orden que en el capítulo anterior, por lo que podremos aplicarles directamente la función POLI y las restantes que vimos en dicho capítulo.

```
[0]   Z←W AJUSTE V
[1]     Z←W⌹V∘.*0,⍳N
```

Programa 29.

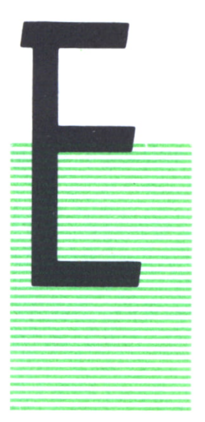
La función tiene una variable global, N, que es el grado del polinomio deseado. Veamos un ejemplo de su utilización:

```
      X←1 2 3 4
      Y←1+(X*3)
      Y
2 9 28 65
      N←2
      Y AJUSTE X
11.5 ¯16.7 7.5
      (Y AJUSTE X) POLI1 X
2.3 8.1 28.9 64.7
      N←3
      Y AJUSTE X
1 0 0 1
      (Y AJUSTE X) POLI1 X
2 9 28 65
```

Programa 30.

En el ejemplo, Y es igual al cubo de X más 1. Por eso, el ajuste polinómico de grado 3 produce los resultados exactos, mientras que el de grado 2 tan sólo da lugar a resultados aproximados. El polinomio, en este caso, es $11.5-16.7\ X+7.5\ X^2$.

NUMEROS COMPLEJOS 6



EL campo de los números complejos es, con respecto al de los números reales, como el plano respecto a la recta. Pues, así como cualquier punto del plano puede representarse por dos coordenadas, es decir, por dos puntos sobre dos rectas privilegiadas (los ejes de coordenadas), todo número complejo puede representarse por un par de números reales (su forma binómica). De hecho, los matemáticos establecen una correspondencia entre los números complejos y los puntos del plano (un isomorfismo) y utilizan unos para representar los otros, y viceversa.

En general, un número complejo puede representarse en forma binómica de la siguiente forma:

$$a + bi$$

donde «a» (parte real del complejo) y «b» (parte imaginaria del complejo) son dos números reales, y donde «i» es la «unidad imaginaria». En la correspondencia con la geometría plana, «a» es la coordenada horizontal, mientras que «b» es la coordenada vertical, que se mide sobre el «eje imaginario».

En APL, representaremos un número complejo como un par de números reales. Por ejemplo (3 + 4i), se representará como el par (3 4).



NOTACION POLAR

La posición de un punto del plano puede representarse de dos maneras: en coordenadas cartesianas (como hemos visto anteriormente) o en coordenadas polares, mediante su módulo (la distancia del punto al origen de coordenadas) y su argumento (el ángulo que forman la recta que le une con el origen y el eje horizontal). En muchos casos es conveniente

pasar de una notación a la otra. Puesto que un número complejo representa un punto del plano en coordenadas rectangulares, también podrá expresarse en notación polar.

Sea a + bi un complejo en notación binomia y r,α el mismo complejo en notación polar (r es el módulo y α el argumento). Las ecuaciones de equivalencia entre las dos formas son:

$$r = (a^2 + b^2)^{0.5}$$
$$\alpha = \text{arc tg}(b/a)$$

$$a = r.\cos \alpha$$
$$b = r.\text{sen } \alpha$$

Veamos un par de funciones APL que transforman un sistema de representación al otro:

```
[0]   Z←POLAR X;M;ALFA
[1]    M←(+/X*2)*0.5
[2]    ALFA←¯3○X[2]÷X[1]
[3]    Z←M,ALFA
```

Programa 1.

Esta función calcula las dos coordenadas polares (módulo y argumento correspondientes a un número complejo expresado en notación binomia, aplicando las dos primeras fórmulas anteriores. La operación APL representada por un círculo con el número –3 a la izquierda representa el arco tangente. (Recuérdese que el mismo círculo con un 3 a la izquierda calcula la tangente.)

```
[0]   Z←BINOMIA P
[1]    Z←P[1]×2 1○P[2]
```

Programa 2.

Esta segunda función realiza la conversión inversa, de la forma polar a la binomia. Es aún más compacta que la anterior. Suponemos que P[1] es el módulo y P[2] el argumento. La operación círculo con los números 2 y 1 a la izquierda calcula a la vez el coseno y el seno de P[2] (del ángulo), pues el 2 representa el coseno y el 1 el seno. Si los multiplicamos por el módulo, obtenemos directamente las dos coordenadas de la forma binomia en el orden adecuado.

Veamos algunos ejemplos de la aplicación de ambas funciones, recordando que los ángulos estarán siempre expresados en radianes:

```
        POLAR 4 4
5.656854249 0.7853981634
        POLAR 3 4
5 0.927295218
        BINOMIA 5.656854249 0.7853981634
4 4
        BINOMIA 5 0.927295218
3 4
```

Programa 3.

Veamos ahora otra versión de las dos funciones anteriores, donde vamos a suponer que los ángulos están expresados en grados, por lo que habrá que convertirlos a radianes al entrar en la función BINOMIA y de radianes a grados al salir de la función POLAR.

```
[0]   Z←POLAR1 X;M;ALFA
[1]    M←(+/X*2)*0.5
[2]    ALFA←¯3○X[2]÷X[1]
[3]    Z←M,ALFA×180÷○1

[0]   Z←BINOMIA1 P
[1]    P[2]←○P[2]÷180
[2]    Z←P[1]×2 1○P[2]
```

Programa 4.

Para pasar de grados a radianes basta multiplicar por "pi" y dividir por 180. Para pasar de radianes a grados, hay que multiplicar por 180 y dividir por «pi». La forma de hacerlo es evidente si recordamos que, en APL, la operación representada por un círculo sin ningún número a la izquierda multiplica por «pi» el valor que se coloca a su derecha. Por tanto, «círculo 1» es idéntico a «pi».

Veamos nuevamente los ejemplos anteriores:

```
        POLAR1 4 4
5.656854249 45
        POLAR1 3 4
5 53.13010235
        BINOMIA1 5.656854249 45
4 4
        BINOMIA1 5 53.13010235
3 4
```

Programa 5.

Vemos que el número binario 4 4 tiene un módulo igual a cuatro veces la raíz de 2 y un ángulo de 45°, mientras que 3 4 tiene un módulo igual a 5 y un ángulo de poco más de 53°.



OPERACIONES CON COMPLEJOS

Tal como hemos representado los números complejos en APL, las operaciones de suma y resta se representan exactamente igual que si se tratara de números ordinarios:

```
          4 4+3 4
7 8
          4 4-3 4
1 0
```

Programa 6.

Para la multiplicación, vamos a construir dos funciones APL. Una que multiplica dos números en notación binómica, y la otra que supone que están en notación polar:

```
[0]   Z←X MULT_C_B Y
[1]     Z←(-/X×Y),(+/X×⌽Y)

[0]   Z←X MULT_C_P Y
[1]  Z←(X[1]×Y[1]),(X[2]+Y[2])
```

Programa 7.

La primera función hace uso del hecho de que la parte real del producto de dos complejos en notación binómica es igual a la diferencia de los productos de las partes reales e imaginarias de ambos, mientras que la parte imaginaria del producto es igual a la suma de los productos alternados. La segunda función, aplicable a la notación polar, se apoya en el hecho de que el módulo del producto es el producto de los módulos y el argumento del producto es la suma de los argumentos de los dos complejos que multiplicamos.

Veamos algunos ejemplos:

```
          4 4 MULT_C_B 3 4
¯4 28
```

```
      BINOMIA (POLAR 4 4) MULT_C_P (POLAR 3 4)
¯4 28
      0 1 MULT_C_B 0 1
¯1 0
```

Programa 8.

En el primer ejemplo hemos multiplicado directamente los dos números en notación binomia (4 4) y (3 4). En el segundo los hemos pasado primero a polares, los hemos multiplicado utilizando la segunda función y hemos convertido el resultado otra vez a la notación binomia. Como es natural, es idéntico al anterior. En el tercer ejemplo hemos calculado i × i, que sale igual a −1. Por eso se dice que la unidad imaginaria es la raíz cuadrada de −1.

Para la división, vamos a definir tan sólo la función APL que trabaja en coordenadas polares, que se basa en el hecho de que el módulo del cociente es el cociente de los módulos, mientras el argumento es igual a la diferencia de los argumentos.

```
[0]  Z←X DIV_C_P Y
[1] Z←(X[1]÷Y[1]),(X[2]-Y[2])
```

Programa 9.

Veamos un ejemplo:

```
      BINOMIA (POLAR 4 4) DIV_C_P (POLAR 3 4)
 1.12 ¯0.16
```

Programa 10.

Finalmente, veamos una función que eleva un número complejo expresado en forma polar a un exponente real. El módulo del resultado es igual al módulo del complejo elevado al exponente, mientras el argumento del resultado es igual al producto del argumento del complejo por el exponente.

```
[0]  Z←X POT_C_P Y
[1] Z←(X[1]*Y[1]),(X[2]×Y[2])
```

Programa 11.

Veamos algunos ejemplos:

```
          BINOMIA (POLAR 4 4) POT_C_P 2
1.960227958E¯15 32
          4 4 MULT_C_B 4 4
0 32
          BINOMIA (POLAR 4 3) POT_C_P 3
¯44 117
          (4 3 MULT_C_B 4 3) MULT_C_B 4 3
¯44 117
```

Programa 12.

Se observará que el cuadrado de un complejo es igual a su producto por sí mismo y el cubo a su producto por sí mismo tres veces. La pequeña discrepancia existente en el cuadrado de (4 4) se debe a los errores de truncación de los números reales en todo ordenador, que son especialmente visibles cuando los resultados de los cálculos se aproximan mucho a cero.

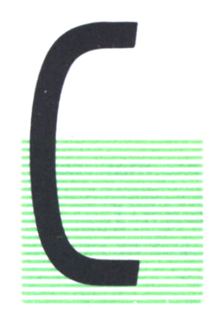
No es necesario hacer una función especial para calcular la raíz cuadrada de un complejo, pues basta usar la función anterior con exponente 0.5:

```
          4 3 MULT_C_B 4 3
7 24
          BINOMIA (POLAR 7 24) POT_C_P 0.5
4 3
```

Programa 13.

Las funciones anteriores son válidas para todos los números complejos comprendidos en el primer cuadrante. Pueden fallar (y precisan correcciones en las que no vamos a entrar aquí, para no complicar los ejemplos) si los números se encuentran exactamente en el eje imaginario o tienen componentes negativas.

DERIVADAS 7



OMO se sabe, se llama «derivada» de una función f(x) en el punto de abscisa x_0 al límite, cuando h tiende a cero, de la expresión

$$\frac{f(x_0 + h) - f(x_0)}{h}$$

Si hacemos variar x_0 a lo largo del eje de abscisas, obtendremos, para cada valor concreto, un valor de la derivada. El conjunto de estos valores forma una nueva función f'(x), que llamamos función derivada de f(x).

Si tenemos una función de dos variables, f(x,y), es posible calcular su derivada con respecto a x, que, en general, será distinta de la derivada con respecto a y.

Cuando la función f(x) es una función algebraica o trigonométrica de x, su derivada respecto a x puede obtenerse de manera muy sencilla, aplicando un conjunto de reglas de derivación, entre las que destacan las siguientes:

1. La derivada de una constante con respecto a cualquier variable independiente es cero.
2. La derivada de una variable independiente y, con respecto a una variable independiente x distinta de y es cero.
3. La derivada de x respecto a x es 1.
4. La derivada de una suma es igual a la suma de las derivadas de sus términos.
5. La derivada de una diferencia es igual a la diferencia de las derivadas de sus términos.
6. La derivada de –f(x) es igual a –f'(x).
7. La derivada del producto de una constante por una función es igual a la constante por la derivada de la función.

8. La derivada de un producto de dos factores es igual a la suma de los productos de cada uno de los dos factores por la derivada del otro. Es decir, si f(x) = u(x).v(x), la derivada de f(x) será igual a:

$$f'(x) = u(x).v'(x) + v(x).u'(x)$$

9. La derivada de la potencia enésima de una función (es decir, de f(x) elevado a *n*, donde *n* es una constante) es igual a *n* por f(x) elevado a n-1 y por f'(x).

10. La derivada del logaritmo de una función es igual a la inversa de la función multiplicada por la derivada de la función.

11. La derivada del seno de una función es igual al coseno de la función multiplicado por la derivada de la función.

12. La derivada del coseno de una función es igual a menos el seno de la función multiplicado por la derivada de la función.

Las cuatro últimas reglas pueden abreviarse de la siguiente manera:

- La derivada de *x* elevado a *n* es igual a *n* por *x* elevado a n−1.
- La derivada del logaritmo de *x* es igual a uno dividido por *x*.
- La derivada del seno de *x* es el coseno de *x*.
- La derivada del coseno de *x* es igual a menos el seno de *x*.
- La derivada con respecto a *x* de una función de función, f(g(x)) es igual a la derivada de f(g(x)) respecto a g(x), multiplicada por la derivada de g(x) con respecto a *x*. Esta regla de derivación se llama «regla de la cadena», pues puede aplicarse tantas veces como se quiera para calcular la derivada de f(g(..(h(x))..))



DERIVADOR SIMBOLICO EN APL

El programa presentado en este apartado está constituido por un conjunto de funciones APL capaces de obtener la derivada de una función algebraica o trigonométrica, expresada en forma simbólica como una cadena de caracteres, que no incluya paréntesis. Una de estas funciones (D) calcula la derivada. Las restantes funciones son auxiliares y son utilizadas por D cuando las necesita.

Este programa no es perfecto, pues su misión es puramente didáctica y no es el objeto de este libro presentar una aplicación totalmente terminada, sino tan sólo abrir caminos para que el lector interesado en el lenguaje APL descubra por sí mismo la inmensa potencia del lenguaje. Por esta razón, hay ciertas funciones algebraicas o trigonométricas cuya derivada el programa no puede calcular. En general, en estos casos no se producirá una derivada errónea (lo que induciría a engaño a quien lo utilice) sino que el propio programa generará un resultado donde aparece la palabra ERROR en el lugar de la derivada que no ha sabido calcular. Esto se

aplica, por ejemplo, a la derivada de cocientes cuyo denominador esté elevado a un exponente (este caso puede obtenerse convirtiendo el cociente en producto y cambiando el signo del exponente del denominador) o a la suma de dos funciones trigonométricas. Aunque también puede aparecer la palabra error si el utilizador se ha equivocado al escribir la función a derivar (recuérdese que el signo de multiplicación debe escribirse siempre).

También deseo avisar aquí de que la derivada de la función cambio de signo (representada por el signo menos a la izquierda de una expresión) puede dar lugar en ciertos casos a resultados erróneos, que en general serán fáciles de localizar. No he tratado de resolver este problema por no complicar el programa y hacerlo más fácil de describir.

Veamos, en primer lugar, el programa completo:

```
⍝ DERIVADOR
[0]   Z←X D FX;U;V;C
[1]     →LOG SI 'LOG('∧.=4↑FX
[2]     →SEN SI 'SEN('∧.=4↑FX
[3]     →COS SI 'COS('∧.=4↑FX
[4]   OP:→(MAS,MENOS,POR,DIV,POT)SI '+-×÷*'∊FX←,FX
[5]     →CONST SI ES_CONST FX
[6]     →ERROR SI 1≠⍴FX
[7]     Z←⍕FX=X
[8]     →0
[9]   CONST:Z←'0'
[10]    →0
[11]  MAS:Z←(X D '+' CABEZA FX)MAS1(X D '+' COLA FX)
[12]    →0
[13]  MENOS:→NEG SI FX[1]='-'
[14]    Z←(X D '-' CABEZA FX)MENOS1(X D '-' COLA FX)
[15]    →0
[16]  NEG:Z←'-',X D 1↓FX
[17]    →0
[18]  POR:V←'×' COLA FX
[19]    →CPOR SI ES_CONST U←'×' CABEZA FX
[20]    Z←(U POR1(X D V))MAS1(V POR1(X D U))
[21]    →0
[22]  CPOR:Z←U POR1 X D V
[23]    →0
[24]  DIV:Z←X D('÷' CABEZA FX)POR1('÷' COLA FX),'*¯1'
[25]    →0
[26]  POT:→ERROR SI~ES_CONST C←'*' COLA FX
[27]    Z←C POR1(X D U)POR1(U←'*' CABEZA FX)POT1⍕¯1+⍎C
[28]    →0
[29]  LOG:→OP SI ')'≠¯1↑FX
[30]    Z←(('(',U,')')POT1 '¯1')POR1 X D U←4↓¯1↓FX
[31]    →0
```

```
[32] SEN:→OP SI ')'≠¯1↑FX
[33]  Z←('COS(',U,')')POR1 X D U←4↓¯1↓FX
[34]  →0
[35] COS:→OP SI ')'≠¯1↑FX
[36]  Z←('(-SEN(',U,'))')POR1 X D U←4↓¯1↓FX
[37]  →0
[38] ERROR:Z←'ERROR'

⍝ SUMA SIMPLIFICADA
[0]  Z←A MAS1 B
[1]   →ACONS SI ES_CONST A
[2]   →BCONS SI ES_CONST B
[3]   →NORED SI∼A EQU B
[4]   Z←'2x',A
[5]   →0
[6]  ACONS:→ABCONS SI ES_CONST B
[7]   →NORED SI 0≠⍎A
[8]   Z←B
[9]   →0
[10] BCONS:→NORED SI 0≠⍎B
[11]  Z←A
[12]  →0
[13] ABCONS:Z←⍕(⍎A)+⍎B
[14]  →0
[15] NORED:Z←A,'+',B

⍝ RESTA SIMPLIFICADA
[0]  Z←A MENOS1 B
[1]   →ACONS SI ES_CONST A
[2]   →BCONS SI ES_CONST B
[3]   →NORED SI∼A EQU B
[4]   Z←'0'
[5]   →0
[6]  ACONS:→ABCONS SI ES_CONST B
[7]   →NORED SI 0≠⍎A
[8]   Z←'-',B
[9]   →0
[10] BCONS:→NORED SI 0≠⍎B
[11]  Z←A
[12]  →0
[13] ABCONS:Z←⍕(⍎A)-⍎B
[14]  →0
[15] NORED:Z←A,'-',B

⍝ MULTIPLICACION SIMPLIFICADA
[0]  Z←A POR1 B
[1]   →ACONS SI ES_CONST A
[2]   →BCONS SI ES_CONST B
[3]   →NORED SI∼A EQU B
```

```
[4]    Z←A,'*2'
[5]    →0
[6]   ACONS:→ABCONS SI ES_CONST B
[7]    →CERO SI 0=⍎A
[8]    →NORED SI 1≠⍎A
[9]    Z←B
[10]   →0
[11] BCONS:→CERO SI 0=⍎B
[12]   →NORED SI 1≠⍎B
[13]   Z←A
[14]   →0
[15] ABCONS:Z←⍕(⍎A)×⍎B
[16]   →0
[17] CERO:Z←'0'
[18]   →0
[19] NORED:Z←A,'×',B
⍝ POTENCIA SIMPLIFICADA
[0]   Z←A POT1 B
[1]    →ACONS SI ES_CONST A
[2]    →BCONS SI ES_CONST B
[3]   NORED:Z←A,'*',B
[4]    →0
[5]   ACONS:→ABCONS SI ES_CONST B
[6]    →UNO SI 1=⍎A
[7]    →NORED SI 0≠⍎A
[8]    Z←'0'
[9]    →0
[10] BCONS:→UNO SI 0=⍎B
[11]   →NORED SI 1≠⍎B
[12]   Z←A
[13]   →0
[14] ABCONS:Z←⍕(⍎A)*⍎B
[15]   →0
[16] UNO:Z←'1'

⍝ FUNCIONES AUXILIARES
    ⍝ EXTRAER DE X LO QUE ESTA A LA IZQUIERDA DE N
[0]   Z←N CABEZA X
[1]    Z←(¯1+(X=N)⍳1)↑X

    ⍝ EXTRAER DE X LO QUE ESTA A LA DERECHA DE N
[0]   Z←N COLA X
[1]    Z←((X=N)⍳1)↓X

[0]   Z←A SI B
[1]    Z←B/A

    ⍝ PRUEBA SI X ES CONSTANTE
[0]   Z←ES_CONST X
```

```
[1]    →0 SI 0=Z←∧/X∊'¯0123456789.'
[2]    X←'Z←0' DEA X

     ⍝ EQUIVALENCIA
[0]   Z←X EQU Y
[1]    Z←0
[2]    →0 SI (⍴⍴X)≠⍴⍴Y
[3]    →0 SI (⍴X)≠⍴Y
[4]    →0 SI (,X)≠,Y
[5]    Z←1
```

Programa 1.

El programa se ejecuta invocando la función D (derivada), que tiene dos argumentos. El izquierdo es la variable respecto a la que queremos derivar, escrita entre comillas. El derecho es la función que queremos derivar, escrita igualmente entre comillas. Veamos algunos ejemplos de su uso:

```
      'X' D '0'
0
      'X' D '5'
0
      'X' D 'Y'
0
      'X' D 'X'
1
      'X' D '2×X'
2
      'X' D '2+X'
1
      'X' D 'X*5'
5×X*4
      'X' D 'X*5+3×X*2-X+5'
5×X*4+3×2×X-1
      'X' D 'X×2×X'
X×2+2×X
      'X' D '1÷X'
¯1×X*¯2
      'X' D '2÷X'
2×¯1×X*¯2
      'X' D 'X*2+2÷X'
2×X+2×¯1×X*¯2
      'X' D '3×X*3-5÷X+12'
3×3×X*2-5×¯1×X*¯2
      'X' D '3×X*3-5÷X*2+12'
3×3×X*2-5×ERROR
      'X' D '3×X*3-5×X*¯2+12'
3×3×X*2-5×¯2×X*¯3
```

```
       'X' D 'LOG(X)'
(X)*¯1
       'X' D 'LOG(2+X)'
(2+X)*¯1
       'X' D 'LOG(2×X)'
(2×X)*¯1×2
       'X' D 'LOG(2×X)+2'
(2×X)*¯1×2
       'X' D 'LOG(2×X)+2×X*4'
(2×X)*¯1×2+2×4×X*3
       'X' D 'SEN(X)'
COS(X)
       'X' D 'SEN(2+X)'
COS(2+X)
       'X' D 'SEN(2×X)'
COS(2×X)×2
       'X' D 'SEN(2×X)+2'
COS(2×X)×2
       'X' D 'SEN(2×X)+2×X*4'
COS(2×X)×2+2×4×X*3
       'X' D 'COS(X)'
(-SEN(X))
       'X' D 'COS(2+X)'
(-SEN(2+X))
       'X' D 'COS(2×X)'
(-SEN(2×X))×2
       'X' D 'COS(2×X)+2'
(-SEN(2×X))×2
       'X' D 'COS(2×X)+2×X*4'
(-SEN(2×X))×2+2×4×X*3
       'X' D 'SEN(LOG(X*5))'
COS(LOG(X*5))×(X*5)*¯1×5×X*4
       'X' D 'SEN(COS(LOG(SEN(X+Y))))'
COS(COS(LOG(SEN(X+Y))))×(-SEN(LOG(SEN(X+Y))))×
         (SEN(X+Y))*¯1×COS(X+Y)
       'Y' D 'SEN(COS(LOG(SEN(X+Y))))'
COS(COS(LOG(SEN(X+Y))))×(-SEN(LOG(SEN(X+Y))))×
         (SEN(X+Y))*¯1×COS(X+Y)
       'Z' D 'SEN(COS(LOG(SEN(X+Y))))'
0
       'X' D '2×COS(LOG(2×X))+2×X'
2×(-SEN(LOG(2×X)))×(2×X)*¯1×2+2
       'X' D '3×X*3-2×X×Y+4×Y*2'
3×3×X*2-2×Y
       'Y' D '3×X*3-2×X×Y+4×Y*2'
-2×X+4×2×Y
       'X' D 'X×COS(X)'
X×(-SEN(X))+COS(X)
```

Programa 2.

Explicación

Comencemos por el programa principal (D), que realiza la derivación simbólica. Es una función de dos variables: la izquierda (X) es la variable independiente respecto a la cual vamos a derivar, mientras que la derecha (FX) es la función a derivar. Ambas variables son literales y contienen una cadena de caracteres que, en el caso de X, debe estar formada por una sola letra (el nombre de la variable).

Este programa consta de 38 líneas que pueden agruparse de la siguiente forma:

— Líneas 1 a 6: distribución de control según la operación a derivar. La línea 1 salta a la instrucción de etiqueta LOG si los cuatro primeros caracteres de la expresión a derivar coinciden con 'LOG('. De igual manera, las líneas 2 y 3 buscan las funciones seno y coseno y transfieren control a las instrucciones de etiquetas SEN y COS si se las encuentra.

La línea 4 busca, a la vez, las operaciones de suma, resta, multiplicación, división y elevación a potencia. En caso de encontrar una suma (máxima prioridad, por estar en primer lugar) el control pasa a la instrucción de etiqueta MAS. Si no había ninguna suma, pero sí una resta, pasa a ejecutarse la instrucción de etiqueta MENOS. Igual ocurre con la multiplicación, la división y la potenciación (etiquetas POR, DIV, POT). El orden en que se han dado las operaciones es el adecuado para que la derivación se efectúe de acuerdo con la prioridad de las operaciones utilizada en la notación ordinaria que se emplea en Matemáticas (no confundir con la prioridad APL, que es exclusivamente posicional).

La línea 5 descubre si la expresión a derivar es una constante (utilizando para ellos la función ES_CONST). En caso afirmativo, transfiere el control a la instrucción de etiqueta CONST. Finalmente, si el programa llega a la línea 6, lo único que puede contener la expresión es una variable aislada, por lo que su longitud debe ser igual a 1. Si no lo es, transferimos control a la etiqueta ERROR.

— Las líneas 7 y 8 analizan el caso de la derivada de una variable con respecto a otra. Si las dos variables coinciden, la derivada es 1. En caso contrario, la derivada es cero. La línea 7 calcula la derivada correctamente en ambos casos y la línea 8 da por terminado el programa.

— Las líneas 9 y 10 (etiqueta CONST) calculan la derivada de una constante, que es cero.

— Las líneas 11 y 12 calculan la derivada de una suma, que es igual a la suma de las derivadas de los dos sumandos. La función auxiliar CABEZA extrae de su argumento derecho (una cadena de caracteres) todo lo que esté a la izquierda del carácter que se le pasa en el argumento izquierdo (en este caso, todo lo que esté a la izquierda del signo '+'). La función

COLA es equivalente, pero extrae todo lo que esté a la derecha de dicho símbolo. Por ejemplo: si FX es igual a 'X*2+3', el resultado de '+' CABEZA FX será 'X*2' (lo que está a la izquierda del signo '+' en FX), mientras que el resultado de '+' COLA FX será '3' (lo que está a la derecha del signo '+' en FX). Estas funciones se utilizan también en líneas subsiguientes.

— Las líneas 13 a 17 calculan la derivada cuando se ha encontrado un signo '-' en la expresión a derivar. En primer lugar, la línea 13 comprueba si se trata de una resta ('-' diádico) o de un cambio de signo ('-' monádico). En este último caso, el signo '-' deberá estar en primer lugar de la cadena. Si es una resta, reciben control las líneas 14 y 15, que calculan la derivada como la diferencia de las derivadas del minuendo y del sustraendo. Si es un cambio de signo, las líneas 16 y 17 calculan la derivada de -F como la derivada de F cambiada de signo.

— Las líneas 18 a 23 calculan la derivada de un producto. La líneas 18 y 19 extraen el multiplicando (que se guarda en la variable U) y el multiplicador (en la variable V) y transfieren control a la etiqueta CPOR si el multiplicando es constante. En caso de que no lo sea, la línea 20 aplica la fórmula de la derivada del producto (U.V'+V.U') y la línea 21 da por terminada la función. La línea 22 actúa si el multiplicando era una constante. En este caso, la derivada es igual a dicha constante por la derivada del multiplicador. La línea 23 termina la ejecución de la función D en este caso.

— La línea 24 calcula la derivada del cociente. Este (U/V) es transformado en producto ($U \times V^{-1}$) cuya derivada se calcula aplicando de nuevo la función D. La línea 25 da por terminada la función.

— Las líneas 26 a 28 calculan la derivada de la potencia. Primero se comprueba que el exponente es constante (si no lo es, se va a ERROR). En caso afirmativo, se calcula la derivada como el producto del exponente por la derivada de la base y por la base elevada al exponente menos 1.

— Las líneas 29 a 31 calculan la derivada del logaritmo. Primero se comprueba que el logaritmo afecta a toda la expresión, comprobando que el paréntesis se cierra al final de ésta (esta comprobación no es perfecta y puede dar lugar a errores que la propia función señalará. Se ha utilizado, a pesar de no ser satisfactoria, por facilitar la explicación del programa. Pero sería casi trivial arreglar este pequeño problema con unas pocas instrucciones más). La derivada de LOG(F), calculada por la línea 30, es igual a la inversa de F (F^{-1}) por la derivada de F.

— De una manera equivalente al caso del logaritmo, las líneas 32 a 34 calculan la derivada del seno, y las líneas 35 a 37 la del coseno.

— Finalmente, si se llega a la instrucción 38, de etiqueta ERROR, se genera el resultado ERROR y se da por terminada la función.

Obsérvese que la función D se llama recursivamente a sí misma mu-

chas veces. Esto es práctica común en APL y viene muy a propósito en este caso, pues simplifica notablemente el proceso.

Veamos ahora someramente las funciones auxiliares:

— MAS1 realiza la suma simplificada de dos cadenas de caracteres. Si una de ellas es cero (el carácter '0', pues estamos trabajando con literales), el resultado es la otra. Si las dos son constantes, realiza la operación y calcula la constante resultante, transformándola de nuevo en caracteres (línea 13). Si las dos son iguales, el resultado es el doble de una de ellas (línea 4). Finalmente, si no se puede aplicar ninguna de las simplificaciones anteriores, el resultado es la concatenación de ambas cadenas, separadas por un signo '+' (línea 15).

— MENOS1 realiza la resta simplificada del mismo modo que MAS1.

— POR1 realiza el producto simplificado. Si una de las expresiones a multiplicar es '0', el resultado será '0' (línea 17). Si una es '1', el resultado será la otra. Si las dos son constantes, realiza la multiplicación (línea 15). Si las dos son iguales, el resultado es una al cuadrado (línea 4). Si no se cumple ninguna de las condiciones anteriores, el resultado es la concatenación de ambas cadenas, separadas por el signo de multiplicar (línea 19).

— POT1 realiza la potenciación simplificada. Si el exponente es '0', el resultado es '1' (línea 16). Si el exponente es '1', el resultado es la base (línea 12). Si la base es '0' o '1', el resultado es la base (líneas 6 a 8). Si la base y el exponente son constantes, realiza la operación (línea 14). Si no se cumple ninguna de las condiciones anteriores, el resultado es la concatenación de ambas cadenas, separadas por un asterisco (línea 3).

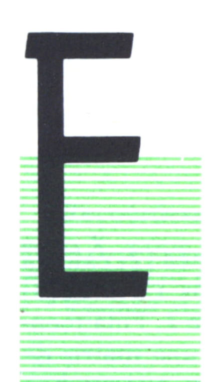
— Ya hemos explicado anteriormente las funciones CABEZA y COLA.

— La función SI permite dar un aspecto más legible a las instrucciones de transferencia condicional.

— La función ES_CONST comprueba si la expresión a la que se aplica es una constante.

— La función EQU comprueba si dos cadenas de caracteres son idénticas. En ese caso, da el resultado 1. En caso contrario, el resultado 0.

ESTADISTICA 8



N este capítulo vamos a ver cómo se programan en APL las operaciones estadísticas más elementales.

SERIES DE DATOS ESTADISTICOS

Para empezar, supongamos que tenemos una serie de datos numéricos como la siguiente:

```
      X←1 3 2 4 3 5 5 6 5 4 3 2 2 1
      X
1 3 2 4 3 5 5 6 5 4 3 2 2 1
```

Programa 1.

En primer lugar, deseamos ordenar dichos datos de menor a mayor. Esto puede lograrse con la siguiente operación APL:

```
      X[⍋X]
1 1 2 2 2 3 3 3 4 4 5 5 5 6
```

Programa 2.

Una operación muy semejante nos los ordena de mayor a menor:

```
      X[⍒X]
6 5 5 5 4 4 3 3 3 2 2 2 1 1
```

Programa 3.



Frecuencias

Veamos ahora con qué frecuencia aparece repetido cada dato en la serie. Para ello bastará utilizar la siguiente función APL:

```
[0]   Z←FRECUENCIAS X;N
[1]     X←X[⍋X]
[2]     N←(¯1↓(0,X)≠X,0)/X
[3]     Z←-1↓(0,X⍳N)-(X⍳N),1+⍴X
[4]     Z←N,[0.5]Z
```

Programa 4.

La línea [1] ordena los datos en orden ascendente. La línea [2] calcula (y guarda en N) los valores distintos que hay en dichos datos. La línea [3] calcula las frecuencias, que quedan guardadas en la variable Z. Finalmente, la línea [4] obtiene el resultado deseado, formado por una matriz de dos filas: la primera contiene los valores distintos de la serie, que estaban en N; la segunda, sus frecuencias respectivas.

Veamos algunos ejemplos de la utilización de esta función:

```
      FRECUENCIAS X
1 2 3 4 5 6
2 3 3 2 3 1
      FRECUENCIAS 0 2 4 6 ¯2 ¯3 2 4 2 2 ¯123 127 4 4 4 4
¯123    ¯3    ¯2     0     2     4     6   127
   1     1     1     1     4     6     1     1
```

Programa 5.



PROMEDIOS

Se llaman promedios las medidas que resumen de alguna manera el conjunto de todos los datos estadísticos de una serie. Existen cinco promedios principales: la media aritmética, la mediana, el promedio típico, la media geométrica y la media armónica.



Media aritmética

La media aritmética de los datos de una serie es igual a la suma de todos ellos dividida por su número. Este es el más utilizado de todos los promedios estadísticos.

La siguiente función APL calcula la media aritmética de una serie de datos. Es una aplicación directa de la definición de media aritmética.

```
[0]  Z←MEDIA X
[1]   Z←(+/X)÷⍴X
```

Programa 6.

Recuérdese que (+/X) significa la suma de todos los elementos de la serie X, y que la operación representada por el símbolo «rho» nos devuelve el número de elementos de la serie a la que se aplica.

Veamos algunos ejemplos de su utilización:

```
      MEDIA ⍳10
5.5
      MEDIA X
3.285714286
```

Programa 7.



Mediana

La mediana se define como el valor que ocupa el lugar central de la serie cuando los elementos de ésta están ordenados de menor a mayor. Si el número de elementos de la serie es impar, no cabe duda de cuál es el valor central, pero si es par, suele tomarse como mediana la media de los dos elementos centrales.

La siguiente función APL calcula la mediana correctamente en los dos casos, tanto cuando el número de elementos de la serie es par, como cuando es impar:

```
[0]  Z←MEDIANA X
[1]   X←X[⍋X]
[2]   Z←+/X[⌈(0 1+⍴X)÷2]÷2
```

Programa 8.

La línea [1] de esta función se limita a ordenar los elementos de la serie de menor a mayor. La línea [2] calcula la media de los dos elementos centrales. Pero el índice de éstos en la serie ordenada está obtenido de tal manera, que si el número de elementos de la serie es impar, nos sale el elemento central repetido dos veces. En este caso, la media de los dos valores obtenidos será igual a uno de ellos, puesto que son iguales.

Veamos algunos ejemplos del cálculo de la mediana:

```
      MEDIANA X
3
      MEDIANA 1 3 5 7 9
5
      MEDIANA 1 3 5 7 9 11
6
```

Programa 9.

Obsérvese que, como la serie X tiene 14 elementos, existen dos valores en lugar central, que en este caso tienen el mismo valor (3, 3), por lo que la mediana es 3. La serie 1 3 5 7 9 tiene cinco elementos, y la mediana nos sale igual a 5 (el que ocupa el lugar central). La serie 1 3 5 7 9 11 tiene seis elementos, y la mediana nos sale igual a la media de los dos centrales (la media de 5 y 7 es 6).



Promedio típico

El promedio típico, también llamado moda, es el valor del elemento de la serie que aparece con más frecuencia. Puesto que ya disponemos de la función FRECUENCIAS, su cálculo es trivial. Pero obsérvese que puede haber más de una moda en una serie de datos, pues podría haber dos o más valores con la misma frecuencia máxima. La función que presento aquí obtiene todas las modas de una serie simultáneamente.

```
[0]  Z←MODA X;I
[1]    Z←FRECUENCIAS X
[2]    I←Z[2;]=⌈/Z[2;]
[3]    Z←I/Z[1;]
```

Programa 10.

Veamos cómo funciona. La línea [1] calcula las frecuencias como una tabla de dos filas (la primera, valores ordenados; la segunda, frecuencias) y las guarda en la variable Z. La línea [2] localiza el valor de la frecuencia

máxima y calcula una serie de ceros y unos que nos dice si cada uno de los valores de dichas frecuencias (la segunda fila de Z) es máximo o no. Esta serie queda guardada en la variable I. Finalmente, la línea [3] obtiene las modas seleccionando de la primera fila de Z los valores correspondientes a los unos de I.

Veamos algunos ejemplos del uso de esta función:

```
          FRECUENCIAS X
  1 2 3 4 5 6
  2 3 3 2 3 1
          MODA X
2 3 5
          Y←1 2 3 4 5 6 5 4 3 2 3 3 3 3
          Y
1 2 3 4 5 6 5 4 3 2 3 3 3 3
          FRECUENCIAS Y
  1 2 3 4 5 6
  1 2 6 2 2 1
          MODA Y
3
```

Programa 11.



Media geométrica

La media geométrica es la raíz N del producto de todos los términos de una serie estadística, donde N es el número de términos de la serie. La siguiente función APL calcula la media geométrica de una serie cualquiera:

```
[0]   Z←MEDIA_G X
[1]    Z←(×/X)*(1÷⍴X)
```

Programa 12.

Veamos algunos ejemplos:

```
          MEDIA_G X
2.871552663
          MEDIA_G Y
3.082680271
```

Programa 13.

Media armónica

La media armónica se define como la inversa de la media aritmética de los inversos de los elementos de la serie. Como ya tenemos la media aritmética en forma de función APL, el cálculo de la media armónica nos resulta trivial, aplicando directamente la definición:

```
[0]  Z←MEDIA_A X
[1]    Z←1÷MEDIA 1÷X
```

Programa 14.

Veamos algunos ejemplos:

```
              MEDIA_A X
      2.427745665
              MEDIA_A Y
      2.763157895
```

Programa 15.

Comparación de los distintos promedios

Para hacernos una idea del valor relativo de los distintos promedios de una misma serie, vamos a ver aquí, conjuntamente, todos los promedios de la serie X, que hemos venido utilizando como ejemplo durante todo este capítulo:

```
      X
1 3 2 4 3 5 5 6 5 4 3 2 2 1
      FRECUENCIAS X
 1 2 3 4 5 6
 2 3 3 2 3 1
      MEDIA X
3.285714286
      MEDIANA X
3
      MODA X
2 3 5
      MEDIA_G X
2.871552663
      MEDIA_A X
2.427745665
```

Programa 16.



MEDIDAS DE DISPERSION

Si los promedios nos dan una medida del «punto medio» de una serie estadística, las medidas de dispersión nos indican hasta qué punto difieren los elementos de la serie de dicho punto medio. Porque las dos series 1 3 5 7 9 y 3 4 5 6 7 tienen la misma media aritmética (5), pero los elementos de la primera están mucho más separados de dicha media que los de la segunda. El conjunto de ambas medidas (promedio y medida de dispersión) da una idea más clara de la distribución de la serie. Vamos a ver aquí dos de las medidas de dispersión principales: la desviación media y la desviación típica.



Desviación media

La desviación media de una serie respecto a un valor arbitrario se define como la suma de los valores absolutos de las diferencias de todos los términos de la serie respecto a dicho valor arbitrario, dividida por el número de términos de la serie.

Normalmente se suele calcular la desviación media respecto a la media aritmética de la serie. Esto es, precisamente, lo que hace la siguiente función APL:

```
[0]  Z←DESV_MEDIA X
[1]   Z←(+/|X-MEDIA X)÷⍴X
```

Programa 17.

La barra vertical utilizada en esta función calcula el valor absoluto de todos los términos de la serie a la que se aplica, que no es otra cosa que la serie de las diferencias con la media aritmética.

Veamos algunos ejemplos:

```
        DESV_MEDIA X
1.326530612
        DESV_MEDIA Y
1.030612245
```

Programa 18.

Obsérvese que la desviación media puede considerarse también como la media aritmética de la serie formada por los valores absolutos de las des-

viaciones de cada término de la serie original con respecto a su media aritmética. Por tanto, la siguiente función APL calculará también correctamente el valor de la desviación media:

```
[0]  Z←DESV_MEDIA X
[1]   Z←MEDIA(|X-MEDIA X)
```

Programa 19.



Desviación típica

Se define la desviación típica como la raíz cuadrada de la media aritmética de los cuadrados de las desviaciones de los términos de la serie con respecto a su media aritmética. Veamos una función APL que la calcula aplicando directamente la definición:

```
[0]  Z←DESV_TIPICA X
[1]   Z←(MEDIA(X-MEDIA X)*2)*0.5
```

Programa 20.

Veamos también algunos ejemplos de su uso:

```
          DESV_TIPICA X
    1.531972185
          DESV_TIPICA Y
    1.287696884
```

Programa 21.



GRAFICOS ESTADISTICOS

Uno de los gráficos estadísticos más frecuentes es el histograma o diagrama de barras verticales. Como ejemplo de la enorme potencia del lenguaje APL, veamos una función de una sola línea que obtiene directamente el histograma de la serie que se le aplica. Esta función se debe a Kenneth Iverson, inventor del lenguaje APL:

```
[0]  HISTO X
[1]    ' *'[1+(11-⍳10)∘.≤⌊0.5+10×(1+X-⌊/X)÷(⌈/X)+1-⌊/X]
```

Programa 22.

Veamos cómo puede utilizarse esta función para obtener el histograma de la distribución de frecuencias de los valores de la serie X, con la que hemos venido trabajando durante todo este capítulo:

```
                    HISTO (FRECUENCIAS X)[2;]
                 ** *
                 ** *
                 ** *
                *****
                *****
                *****
                *****
                ******
                ******
                ******
```

Programa 23.

Una modificación mínima de la función anterior nos permite obtener el gráfico lineal de cualquier función, siempre que los valores de la variable independiente sean consecutivos:

```
[0]  GRAF X
[1]    ' *'[1+(11-⍳10)∘.=⌊0.5+10×(1+X-⌊/X)÷(⌈/X)+1-⌊/X]
```

Programa 24.

Veamos un ejemplo: supongamos que la variable independiente X toma los valores de 1 a 8 y que la función que vamos a representar es igual a (X-3) multiplicado por (X-6), es decir, el polinomio X^2 -9X+18. Aplicando a esta función el programa GRAF, podemos ver fácilmente que la gráfica correspondiente es una parábola.

```
      X←ι8
      Y←(X-3)x(X-6)
      GRAF Y
*      *

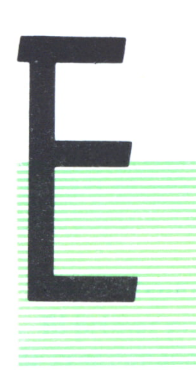



 *    *


  *  *
   **
```

Programa 25.

GEOMETRIA 9



L lenguaje APL es especialmente apto para el cálculo de las fórmulas elementales de la Geometría. La mayor parte de las operaciones geométricas pueden transformarse a expresiones de este lenguaje de forma casi inmediata. Pero donde se observa mayor facilidad de trabajo es cuando se utilizan coordenadas rectangulares en el plano o en el espacio.

FORMULAS GEOMETRICAS CLASICAS

En primer lugar, vamos a ver cómo se calculan en el lenguaje APL diversas expresiones geométricas clásicas. Comenzaremos por el área del círculo, dado su radio. La siguiente función APL lo calcula en función del radio:

```
[0]   Z←AREA_C R
[1]     Z←○R*2
```

Programa 1.

Como se sabe, la operación APL representada por un círculo multiplica por «PI» el valor que aparece a su derecha, que en este caso es igual al cuadrado del radio R. El resultado obtenido es, por tanto, «PI por R al cuadrado», que es la fórmula conocida.

Veamos algunos ejemplos:

```
      AREA_C 1
3.141592654
```

```
          AREA_C 2
12.56637061
          AREA_C 3
28.27433388
```

Programa 2.

La siguiente función calcula la longitud de la circunferencia en función de su radio:

```
[0]   Z←LONG_C R
[1]     Z←2×○R
```

Programa 3.

Su funcionamiento refleja la fórmula clásica de forma inmediata, también en este caso. Veamos algunos ejemplos:

```
          LONG_C 1
6.283185307
          LONG_C 2
12.56637061
          LONG_C 3
18.84955592
```

Programa 4.

Veamos ahora el área del cuadrado, dado el lado, junto con algunos ejemplos:

```
[0]   Z←AREA2 L
[1]     Z←L*2
          AREA2 1
1
          AREA2 2
4
          AREA2 10
100
```

Programa 5.



MEDIDA DE DISTANCIAS

Vamos a representar los conjuntos de puntos como matrices o tablas, donde cada fila representa las coordenadas de un punto. De esta manera, si trabajamos en dos dimensiones, cada punto tendrá dos coordenadas, por lo que el conjunto de puntos vendrá representado por una matriz de dos columnas y tantas filas como puntos haya en el conjunto, como la siguiente:

```
      T2←3 2⍴0 0 3 0 0 4
      T2
0 0
3 0
0 4
```

Programa 6.

El primer punto del conjunto T2 está en el origen de coordenadas, pues sus coordenadas son (0,0). El segundo está sobre el eje de abscisas, pues sus coordenadas son (3,0). Finalmente, el tercer punto está sobre el eje de ordenadas, en el punto de coordenadas (0,4).

De igual manera, un conjunto de puntos en el espacio de tres dimensiones podrá representarse por una matriz de tres columnas y tantas filas como puntos haya en el conjunto. Por ejemplo, la siguiente:

```
      T3←3 3⍴0 0 0 5 5 5 5 0 0
      T3
0 0 0
5 5 5
5 0 0
```

Programa 7.

Se trata, esta vez, de un conjunto de tres puntos con coordenadas espaciales (0,0,0), (5,5,5) y (5,0,0).

De la misma forma podríamos formar un conjunto de puntos en más de tres dimensiones. Por ejemplo, en el espacio de cuatro dimensiones tendríamos que definir una matriz de cuatro columnas y tantas filas como puntos, tal como la siguiente:

```
      T4←3 4⍴0 0 0 5 5 0 0 0 5 0 5
      T4
```

```
0 0 0 0
5 5 0 0
0 5 0 5
```

Programa 8.

Los tres ejemplos anteriores definían conjuntos de tres puntos. Esto no es necesario. Podrían ser cuatro, como en el caso siguiente:

```
          C2←4 2⍴2 2 5 2 5 5 2 5
          C2
0 0
5 2
5 5
2 5
```

Programa 9.

La siguiente función APL calcula la distancia entre cada dos puntos consecutivos de un conjunto de puntos. El número de distancias obtenido coincide con el número de puntos en el conjunto. La última distancia es la que separa el último punto del primero. Es decir, el conjunto se considera cerrado.

```
[0]  Z←DIST P;D
[1]   D←P-1⊖P
[2]   Z←(+/D*2)*0.5
```

Programa 10.

La primera línea de este programa calcula la diferencia entre las coordenadas de cada dos filas consecutivas. La segunda calcula el valor absoluto de las distancias, como raíz cuadrada de la suma de los cuadrados de las diferencias de coordenadas obtenidas por la línea [1].

Veamos cómo se aplica esta función a los conjuntos de puntos definidos anteriormente:

```
      DIST T2
3 5 4
      DIST T3
8.660254038 7.071067812 5
```

```
        DIST T4
7.071067812 7.071067812 7.071067812
        DIST C2
3 3 3 3
```

Programa 11.

Obsérvese que el número de distancias es siempre igual al número de puntos. Que el conjunto de puntos T2 define un triángulo rectángulo de lados iguales a 3, 5 y 4 (pues el cuadrado de la hipotenusa, 25, es igual a la suma de los cuadrados de los catetos, 9 y 16, de acuerdo con el teorema de Pitágoras). También puede verse que C2 define un cuadrado de lado 3.



POLIGONOS

La siguiente función APL comprueba si un conjunto de puntos dado en forma de matriz (como se vio en el apartado anterior) forma un polígono (el resultado de la función será 1) o no puede formarlo (resultado igual a cero). Este último caso ocurrirá, normalmente, si los puntos están alineados.

```
[0]  Z←POLIGONO P;D;SP
[1]   D←DIST P
[2]   SP←0.5×+/D
[3]   Z←∧/D<SP
```

Programa 12.

La primera línea de esta función calcula las distancias entre cada dos puntos consecutivos utilizando la función DIST. La segunda calcula el semiperímetro del supuesto polígono (la mitad de la suma de las distancias). La tercera comprueba que todas las distancias son menores que el semiperímetro, condición indispensable para que se forme polígono.

Apliquemos la función a los conjuntos de puntos anteriores, para ver cuáles forman polígono y cuáles no:

```
        POLIGONO T2
1
        POLIGONO T3
1
```

```
            POLIGONO T4
   1
            POLIGONO C2
   1
```

Programa 13.

Todos ellos forman polígono. Probemos ahora la función con tres puntos alineados: (0,0), (2,2) y (4,4). Los tres se encuentran sobre la bisectriz del primer cuadrante del sistema de coordenadas utilizado:

```
      POLIGONO 3 2⍴0 0 2 2 4 4
0
```

Programa 14.

Efectivamente. Como podíamos esperar, estos tres puntos no forman un polígono. Obsérvese que podemos aplicar la misma función, tanto en el plano, como en el espacio de tres dimensiones, como en espacios matemáticos de más de tres dimensiones.

TRIANGULOS

En primer lugar, vamos a calcular el área de un triángulo por la fórmula de Herón, que dice que dicha área es igual a la raíz cuadrada del producto del semiperímetro del triángulo por las tres diferencias entre dicho semiperímetro y las longitudes de los tres lados. Esta es la función APL aplicable en este caso:

```
[0]   Z←AREA T;D;SP
[1]     D←DIST T
[2]     SP←0.5×+/D
[1]     Z←(×/SP,SP-D)*0.5
```

Programa 15.

Su explicación es inmediata, recordando la función anterior. La línea [3] es la expresión exacta de la fórmula de Herón.

Esta función también puede aplicarse para calcular el área de un triángulo, cualquiera que sea el número de dimensiones del espacio sobre el que esté definido. Veámoslo:

```
        AREA T2
6
        AREA T3
17.67766953
        AREA T4
21.65063509
        AREA 3 2⍴0 0 2 2 4 4
0
```

Programa 16.

Obsérvese que incluso podemos aplicarla a un conjunto de tres puntos alineados, aunque en este caso nos da un área igual a cero, como es lógico.

La siguiente función APL calcula las coordenadas del baricentro de un triángulo definido por las coordenadas de sus vértices:

```
[0]   Z←BCENTRO T
[1]     Z←(+⌿T)÷3
```

Programa 17.

En efecto, el baricentro de un triángulo tiene sus coordenadas iguales a la media aritmética de las coordenadas de los vértices del triángulo. Veamos cómo se aplica:

```
        BCENTRO T2
1 1.333333333
        BCENTRO T3
3.333333333 1.666666667 1.666666667
        BCENTRO T4
1.666666667 3.333333333 0 1.666666667
```

Programa 18.

El programa que vamos a ver a continuación es un caso más general del cálculo del baricentro y puede aplicarse a la resolución de problemas de estática. Supongamos que tenemos varios pesos diferentes situados en los vértices de un triángulo: ¿cuál será el centro de masas del triángulo? La siguiente función lo calcula:

```
[0]   Z←M GCENTRO T
[1]    Z←(M÷+/M)+.×T
```

Programa 19.

Veamos varios ejemplos de su utilización:

```
        1 1 1 GCENTRO T2
1 1.333333333
        2 2 2 GCENTRO T2
1 1.333333333
        1 2 3 GCENTRO T2
1 2
        4 2 2 GCENTRO T2
0.75 1
        2 4 2 GCENTRO T3
3.75 2.5 2.5
        2 4 2 GCENTRO T4
2.5 3.75 0 1.25
```

Programa 20.

Se observará que, si las tres masas son iguales, el centro de masas (o centro de gravedad) coincide con el baricentro del triángulo. Si las masas son diferentes, el centro de masas se desplaza hacia las masas más grandes. La misma función puede aplicarse al plano, al espacio tridimensional y a espacios de más dimensiones, así como a cualquier número de puntos, y no sólo a triángulos.

APROXIMACION DE PI

Vamos a calcular una aproximación de PI, obteniendo el perímetro de un polígono de 2 × N lados inscrito en la circunferencia de diámetro unidad. Como se sabe, la longitud de esta circunferencia es exactamente igual a PI. Para calcular el perímetro del polígono, bastará con multiplicar por 2 su semiperímetro. En cuanto a éste, es el conjunto de distancias entre cada dos vértices del polígono situados por encima del eje de abscisas. Veamos la función APL que realiza este cálculo:

```
[0]   Z←PI N
[1]    N←0,(⍳N)÷N
[2]    Z←2×+/¯1↓DIST N,[1.5]0○N
```

Programa 21.

La primera línea divide el intervalo 0,1 en N partes iguales. En la segunda línea encontramos primero (leyendo de derecha a izquierda) la expresión

```
0○N
```

Programa 22.

que calcula la raíz cuadrada de N cuadrado menos 1 para cada uno de los valores de N. Ahora tenemos la expresión

```
N,[1.5]0○N
```

Programa 23.

que forma una matriz cuya primera columna es N (las abscisas de los vértices) y la segunda es el resultado de la expresión anterior (las ordenadas de los vértices). Ya tenemos, por tanto, calculadas las coordenadas de todos los vértices del polígono inscrito que están por encima del eje de abscisas. Ahora tenemos que calcular la distancia entre cada dos de ellos, para lo que aplicamos la función DIST. Pero como esta función nos cierra el polígono, calculando también la distancia entre el último punto y el primero, que aquí no nos interesa, la eliminamos con la expresión

```
¯1↓DIST N,[1.5]0○N
```

Programa 24.

Finalmente, sumamos todas las distancias obtenidas (lo que nos da el semiperímetro) y multiplicamos por 2 para obtener el perímetro, que será una aproximación de PI tanto mejor cuanto mayor sea el número de lados del polígono.

Veamos algunos ejemplos:

```
      PI 10
3.132264619
      PI 20
3.13830042
      PI 30
```

```
3.139801571
         PI 50
3.14076059
         PI 100
3.141298567
         PI 200
3.141488694
         PI 500
3.141566356
         PI 1000
3.141583356
         PI 1500
3.141587593
```

Programa 25.

Comparemos los resultados anteriores con el valor real de PI:

```
        o1
3.141592654
```

Programa 26.



CONCLUSION

Podrían ponerse muchos más ejemplos de la aplicabilidad del lenguaje APL a la enseñanza de las matemáticas, la estadística o cualquier otra de las ciencias. Mas creo que basta con lo anteriormente expuesto para que el lector pueda hacerse una idea de la enorme potencia de este lenguaje y la facilidad de programación que proporciona. El inventor del lenguaje APL (Ken Iverson) ha publicado varios libros que tratan, precisamente, de su utilidad para la explicación de los conceptos del álgebra, la lógica, el análisis de circuitos eléctricos, la traducción de lenguajes de ordenador y otras muchas áreas. El autor de este libro lleva también cierto tiempo trabajando en el diseño de extensiones al lenguaje APL para hacerlo más aplicable aún a la programación de aplicaciones de inteligencia artificial y de sistemas expertos.

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

INDICE GENERAL

1 COMO CONSTRUIR JUEGOS DE AVENTURA

Descripción y ejemplos de las principales familias de aventura para ordenador: simuladores de combate, aventuras espaciales, búsquedas de tesoros..., terminando con un programa que permite al lector construir sus propios libros de multiaventura.

2 COMO DIBUJAR Y HACER GRAFICOS CON EL ORDENADOR

Desde el primer «brochazo» aprenderá a diseñar y colorear tanto figuras sencillas como las más sofisticadas creaciones que pueda llegar a imaginar, sin necesidad de profundos conocimientos informáticos ni artísticos.

3 PROGRAMACION ESTRUCTURADA EN EL LENGUAJE PASCAL

Invitación a programar en PASCAL, lenguaje de alto nivel que permite programar de forma especialmente bien estructurada, tanto para aquellos que ya han probado otros lenguajes como para los que se inician en la informática.

4 COMO ELEGIR UNA BASE DE DATOS

Libro eminentemente práctico con numerosos cuadros y tablas, útil para poder conocer las bases de datos y elegir la que más se adecúe a nuestras necesidades.

5 AÑADA PERIFERICOS A SU ORDENADOR

Breve descripción de varios periféricos que facilitan la comunicación con el ordenador personal, con algunos ejemplos de fácil construcción: ratón, lápiz óptico, marco para pantalla táctil...

6 GRAFICOS ANIMADOS CON EL ORDENADOR

En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comporender. Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc.

7 JUEGOS INTELIGENTES EN MICROORDENADORES

Los ordenadores pueden enfrentarse de forma «inteligente» ante puzzles y otros tipos de juegos. Esto es posible gracias al nuevo enfoque que ha dado la IA a la tradicional teoría de juegos.

8 PERIFERICOS INTERACTIVOS PARA SU ORDENADOR

Descripción detallada de la forma de construir, paso a paso y en su propia casa, dispositivos electrónicos que aumentarán la potencia y facilidad de uso de su ordenador: tableta digitalizadora, convertidores de señales analógicas, comunicaciones entre ordenadores.

9 COMO HACER DIBUJOS TRIDIMENSIONALES EN EL ORDENADOR PERSONAL

Compruebe que también con su ordenador personal puede llegar a diseñar y calcular imágenes en tres dimensiones con técnicas semejantes a las utilizadas por los profesionales del dibujo con equipos mucho más sofisticados.

10 PRACTIQUE MATEMATICAS Y ESTADISTICA CON EL ORDENADOR

En este libro se repasan los principales conceptos de las Matemáticas y la Estadística, desde un punto de vista eminentemente práctico y para su aplicación al ordenador personal. Se basan los diferentes textos en la presentación de pequeños programas (que usted podrá introducir en su ordenador personal).

11 CRIPTOGRAFIA: LA OCULTACION DE MENSAJES Y EL ORDENADOR

En este libro se presentan las técnicas de mensajes a través de la criptografía desde los primeros tiempos hasta la actualidad, en que el uso de los computadores ha proporcionado la herramienta necesaria para llegar al desarrollo de esta técnica.

12 APL: LENGUAJE PARA PROGRAMADORES DIFERENTES

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas y al mismo tiempo permite programar sin necesidad de conocer todos los elementos del lenguaje. Por ello es ideal para quienes reúnan imaginación y escasa formación en Informática.

13 ECONOMIA DOMESTICA CON EL ORDENADOR PERSONAL

Breve introducción a la contabilidad de doble partida y su aplicación al hogar, con explicaciones de cómo utilizar el ordenador personal para facilitar los cálculos, mediante un programa especialmente diseñado para ello.

14 COMO SIMULAR CIRCUITOS ELECTRONICOS EN EL ORDENADOR

Introducción a los diferentes métodos que se pueden emplear para simular y analizar circuitos electrónicos, mediante la utilización de diferentes lenguajes.

15 COMO CONSTRUIR SU PROPIO ORDENADOR

Cuando se trabaja con un ordenador, lo único que puede apreciarse, a simple vista, es una especie de caja negra que, misteriosamente, acepta una serie de instrucciones. En realidad, un ordenador es una máquina capaz de recibir, transformar, almacenar y suministrar datos.

16 EL ORDENADOR COMO INSTRUMENTO MUSICAL Y DE COMPOSICION

Análisis de cómo se puede utilizar el ordenador para la composición o interpretación de música. Libro eminentemente práctico, con numerosos ejemplos (que usted podrá practicar en su ordenador casero) y lleno de sugerencias para disfrutar haciendo de su ordenador un verdadero instrumento musical.

17 SISTEMAS OPERATIVOS: EL SISTEMA NERVIOSO DEL ORDENADOR

Características de diversos sistemas operativos utilizados en los ordenadores personales y caseros. Se trata de llegar al conocimiento, ameno aunque riguroso, de la misión del sistema operativo de su ordenador, para que usted consiga sacar mayor rendimiento a su equipo.

18 UNIX, EL ESTANDAR DE LOS SISTEMAS OPERATIVOS MULTIUSUARIO

La aparición y posterior difusión del sistema operativo UNIX supuso una revolución en el mercado, de tal modo que se ha convertido en el estándar de los sistemas multiusuario. Su aparente complejidad podría provocar, en principio, un primer rechazo, pero debido a su potencia se convierte rápidamente en una extraordinaria herramienta de trabajo apta para cualquier tipo de aplicaciones.

19 EL ORDENADOR Y LA ASTRONOMIA

Los cálculos astronómicos y el conocimiento del firmamento en un libro apasionante y curioso.

20 VISION ARTIFICIAL. TRATAMIENTO DE IMAGENES POR ORDENADOR

El procesado de imágenes es un campo de reciente y rápido desarrollo con importantes aplicaciones en área tan diversas como la mejora de imágenes biomédicas, robóticas, teledetección y otras aplicaciones industriales y militares. Se presentan los principios básicos, los sistemas y las técnicas de procesado más usuales.

21 PRACTIQUE HISTORIA Y GEOGRAFIA CON SU ORDENADOR

Libro interesante para los aficionados a estas ciencias, a quienes presenta una nueva visión de cómo utilizar el microordenador en su estudio.

22 LA CREATIVIDAD EN EL ORDENADOR. EXPERIENCIAS EN LOGO

El LOGO es un lenguaje enormemente capacitado para la creación principalmente gráfica y en especial para los niños. En este sentido se han desarrollado numerosas experiencias. En el libro se analizan estas experiencias y las posibilidades del LOGO en este sentido, así como su aplicación a su ordenador casero para que usted mismo (o con sus hijos) pueda repetirlas.

23 EL LENGUAJE C, PROXIMO A LA MAQUINA

Lenguaje de programación que se está imponiendo en los microordenadores más grandes, tanto por su facilidad de aprendizaje y uso, como por su enorme potencia y su adecuación a la programación estructurada. Vinculado íntimamente al sistema operativo UNIX es uno de los lenguajes de más futuro entre los que se utilizan los micros personales.

24 BASIC

El lenguaje BASIC es la forma más fácil de aprender las instrucciones más elementales con las que podemos mandar a nuestro ordenador que haga las más diversas tareas.

25 COMO ELEGIR UNA HOJA ELECTRONICA DE CALCULO

En este título se estudian las diferentes versiones existentes de esta aplicación típica, desde el punto de vista de su utilidad para, en función de las necesidades de cada usuario y del ordenador de que dispone, poder elegir aquella que más se adecúe a cada paso.

26 BASIC AVANZADO

Una vez conocidas las instrucciones fundamentales del lenguaje BASIC se plantea la cuestión de la realización de programas que resuelvan problemas o aplicaciones que se nos presentan diariamente en el trabajo, en casa o en los estudios. Este libro trata de mostrar cómo se podrían realizar algunas de estas aplicaciones, estudiando diversas estructuras que proporciona el lenguaje BASIC (como las subrutinas) y viendo las ideas fundamentales para realizar gráficos en pantalla mediante un programa y para almacenar datos en discos o cintas mediante los ficheros.

27 APLIQUE SU ORDENADOR A LAS CIENCIAS NATURALES

Ejemplos sencillos para practicar con el ordenador. Casos curiosos de la Naturaleza en forma de programas para su ordenador personal.

28 PRACTIQUE FISICA CON SU ORDENADOR

Deja que el ordenador te ayude en tus estudios. Materias tan difíciles como la Física, se ponen a tu alcance de una manera entretenida y mucho más clara, con programas que te permitirán entender las cosas desde un punto de vista más real.
Definiciones, fórmulas, gráficos y ejemplos, en un pequeño manual que puedas utilizar en cualquier momento.

29 PRACTIQUE QUIMICA CON SU ORDENADOR

En nuestra búsqueda particular de la «piedra filosofal», al modo de los antiguos alquimistas, nos ayudaremos del ordenador para que nos resulte más fácil. Con este libro conseguiremos entender fácilmente las valencias de los elementos, las reacciones Redox y las distintas teorías sobre el átomo. Nos servirá de guía para aprender la tabla periódica de los elementos y nos ayudará a comprender, mediante gráficos, una reacción en cadena. Podremos así convertir nuestra casa y nuestro ordenador en un gran laboratorio.

30 APRENDA MATEMATICAS Y ESTADISTICA CON EL LENGUAJE APL

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas. Indudablemente, es mucho más apto que BASIC para la construcción de pequeños programas que realicen operaciones matemáticas de dificultad media, que además se expresan de una forma muy semejante a la notación matemática ordinaria, lo que lo hace fácilmente comprensible.

31 LOS LENGUAJES DE LA INTELIGENCIA ARTIFICIAL

Libro en que se describen los lenguajes específicos para la «elaboración del saber» y los entornos de programación correspondientes. El conocimiento de estos lenguajes, además de interesante en sí mismo, es sumamente útil para entender todo lo que la Inteligencia Artificial supondrá para el futuro de la Informática.

32 LA ESTACION TERMINAL PERSONAL

Las modernas técnicas de comunicación van permitiendo que las grandes capacidades de proceso y el acceso a bases de datos de gran tamaño estén cada día más al alcance de cada usuario (fuera ya de los centros de proceso de datos).

33 COBOL

Este libro pretende introducir al lector en uno de los lenguajes más utilizados y menos considerados del mundo informático. El Cobol es el lenguaje de gestión por excelencia y está presente en el desarrollo del software en la gran mayoría de empresas e instituciones públicas.

34 ADA

El considerable esfuerzo desarrollado por el Departamento de Defensa de los Estados Unidos (DoD) para que el lenguaje Ada fuese desarrollado quedará compensado por las aportaciones de este lenguaje a los sistemas informáticos del futuro.
Sus aplicaciones originales, sistemas en tiempo real para mando y control en el área de Defensa, han sido ampliadas al campo industrial para el control de procesos, aplicaciones en tiempo real, inteligencia artificial, etcétera.

35 EL ORDENADOR Y LA LITERATURA

En este libro se examinan procesadores de textos, programas de análisis literario y una curiosa aplicación desarrollada por el autor: APOLO, un programa que compone estructuras poéticas.

36 EL ORDENADOR COMO MAQUINA DE ESCRIBIR INTELIGENTE

Descripción de algunos de los programas para tratamiento de textos existentes en el mercado, análisis comparativos y estudio de las posibilidades de cada uno de ellos. Guía práctica para la elección del procesador de textos que más se adecúe a nuestras necesidades y al ordenador personal del que dispongamos.

37 MS-DOS

El sistema operativo de muchos ordenadores personales es el sistema operativo de disco de Microsoft, más conocido como MS-DOS, que recibe su nombre de su principal actividad: manejar los discos y archivos de discos. Su conocimiento puede llegar a ser tan profundo como deseemos, las nociones básicas, sin embargo, pueden llegar a ser imprescindibles para el manejo de nuestro ordenador.

38 REDES DE AREA LOCAL

El objetivo de este libro es el de proporcionar al lector un conocimiento claro de lo que son las redes locales, de su tecnología, problemática y futuro, de forma que, si lo desea, pueda profundizar posteriormente, por medio de bibliografía especializada o por la práctica profesional.

39 LOS FUNDAMENTOS DE LA GRAFOLOGIA APLICADA Y SU POSIBLE TRATAMIENTO CON UN ORDENADOR PERSONAL

Se presentan en este libro los perfiles grafológicos óptimos correspondientes a diversas actividades laborales, así como los programas de ordenador necesarios para el manejo de estos datos. Obra eminentemente práctica y de aplicación de los conceptos teóricos desarrollados en ella.

40 ¿MAQUINAS MAS EXPERTAS QUE LOS HOMBRES?

Después de situar los «sistemas expertos» en el contexto de la Inteligencia Artificial y describir su construcción, su funcionamiento, su utilidad, etc., se analiza el papel que pueden tener en el futuro (y en el presente, ya) de la Informática, así como los polémicos temas de la «capacidad para desbancar a la inteligencia humana», y las posibilidades de «aprender» de que se puede dotar a un procesador, etcétera.

NOTA:
Ediciones Siglo Cultural, S. A., se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de esta colección.

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas.

Indudablemente es mucho más apto que BASIC para la construcción de pequeños programas que realicen operaciones matemáticas de dificultad media, que además se expresan de una forma muy semejante a la notación matemática ordinaria, lo que los hace fácilmente comprensibles.